Network Server/Consolidator

B. Lublinsky

Fermi National Accelerator Laboratory

Abstract

During the last several years Fermilab's control system was undergoing significant changes. More and more individual subsystems have been taken out of Camac crates and implemented as standalone boxes connected directly to the network. This relieves the Camac Front Ends [1] from controlling those subsystems, but creates significant additional load on the network, and what is even worse, some of the functionality that was previously supported by Camac Front Ends is gone with this approach. A most important feature that was supported through Camac Front Ends is the ability to do "ring-wide" reading and setting (the same "device" around the ring, supported through the different subsystems could be manipulated by one request). The necessity to support this feature, without further abusing the network, forced the creation of the network server/consolidator that is described in this paper.

1. Major system requirements.

The major requirements for this type of system are as follows:
1. It has to be flexible enough to allow adding new nodes and new "associations" between nodes;
2. It has to minimize network traffic that it originates.

The first requirement can be satisfied relatively easily by using multiple node association tables (fig 1). Based on the table number that the request is referring to, internal software builds whatever amount of requests is necessary to the nodes specified in the association table, sends them out, compiles the replies together and sends results to the original requester.

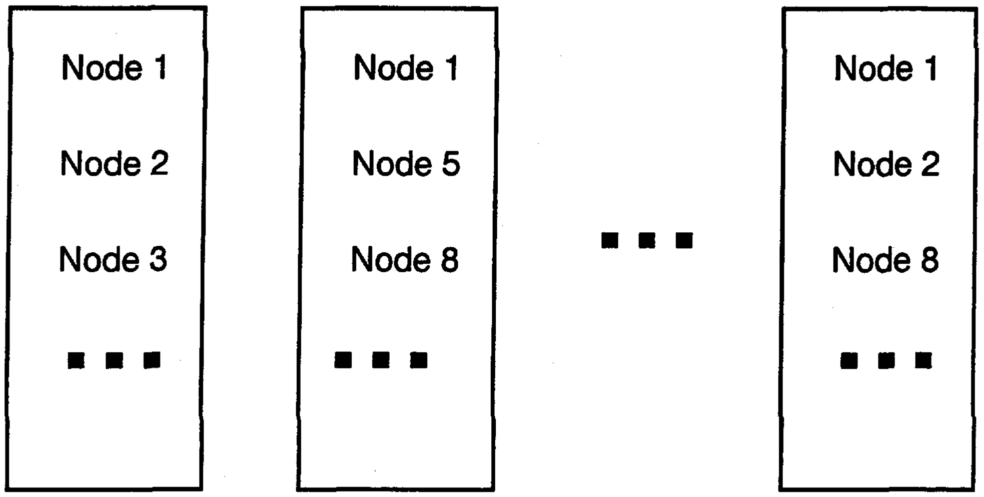


Fig 1. Association tables for Server/consolidator.

The second requirement is significantly tougher. If we examine the techniques that have been used for implementing microprocessor based systems at Fermilab, we will find two distinct approaches. One of them is the approach used in the Camac Front End [1]. Let us call it "no pool" approach. This approach is usually used in the case when machine is capable of front-ending very many devices, but only small part of them are accessed at a time. The advantage of this approach is that the machine is doing only what it has to do to satisfy current requests. The software implementation of this approach is fairly straightforward: you get the request, you go and get the data, you return it back. The obvious drawback is that for duplicated requests the data will be collected multiple times. This approach is usually acceptable if acquiring the necessary data is relatively "cheap" in terms of time and hardware efforts to get it. The second approach is usually used in the dedicated systems with relatively small amount of data [2]. In this case, the whole system usually runs from a fixed data pool that is collecting all system parameters. With organization like this, data is assumed always to be there, so any request from the outside can be satisfied immediately. This implementation usually implies two major levels of data processing: internal data collection that ensures data pool updates with the rate that is dictated by the system requirement, and replies to the rest of the world with the frequency specified by requests. The advantage of this approach is that the data access is always synchronous and straightforward. The biggest drawback is that usually the system is collecting more data that it really needs. But if the amount of this data is relatively small, pool paradigms are usually used.

For the consolidator system neither one of these two approaches will work satisfactorily. Running a fixed data pool is virtually impossible because we don't even know up front what will be the data that we will have to collect, and no pool solution can turn out to be too expensive from the point of view of the network traffic. The solution that has been used for this system is a dynamic pool, the kind of approach that is used on the consoles for request consolidation. When the system starts up, the pool is empty and no data collection is happening. When a request comes in, the first thing that is done is check whether this data is already available through the pool. If it is, then the request will be reusing existing data collection, if not, then new data collection will be started in the pool. The rules for appending of the requested item to the element of the pool are as follows:

1. Data collection frequency of the new request should not exceed data collection frequency of the existing element of the pool;
2. Length of the collected data for the new request should not exceed length of the collected data in the pool.

When all of the requests pointing to the specific element of the pool are canceled, this element of the pool will be deleted. This allows us to take advantage of both approaches described above. Only the data that is currently necessary is collected, every duplicated data collection is merged by the pool. An additional caveat here is that one request can build many requests going to different nodes. Having these small requests going back and forth can degrade significantly network performance. Even worse, this can saturate the network hardware. That is why, besides pooling of the requests themselves, we have to combine outgoing requests to different nodes as much as we can. For every new request we are comparing its reply frequency to the reply frequencies of the requests that already exist for this node. If the frequences are close enough and the combined reply length fits into the ACNET packet, the existing reply is killed. It is then combined with the new request and restarted.

2. Implementation.

Consolidator/Server is implemented using a 68040 VME-based system with two networking interfaces: Token Ring, used for communication with the cryogenic control system [2], and Ethernet, used for communication with the rest of the systems; it utilizes the VX-Works real time kernel. It uses Fermilab's "standard" timing and communication support [3]. The whole system is organized around three major queues (fig 2).

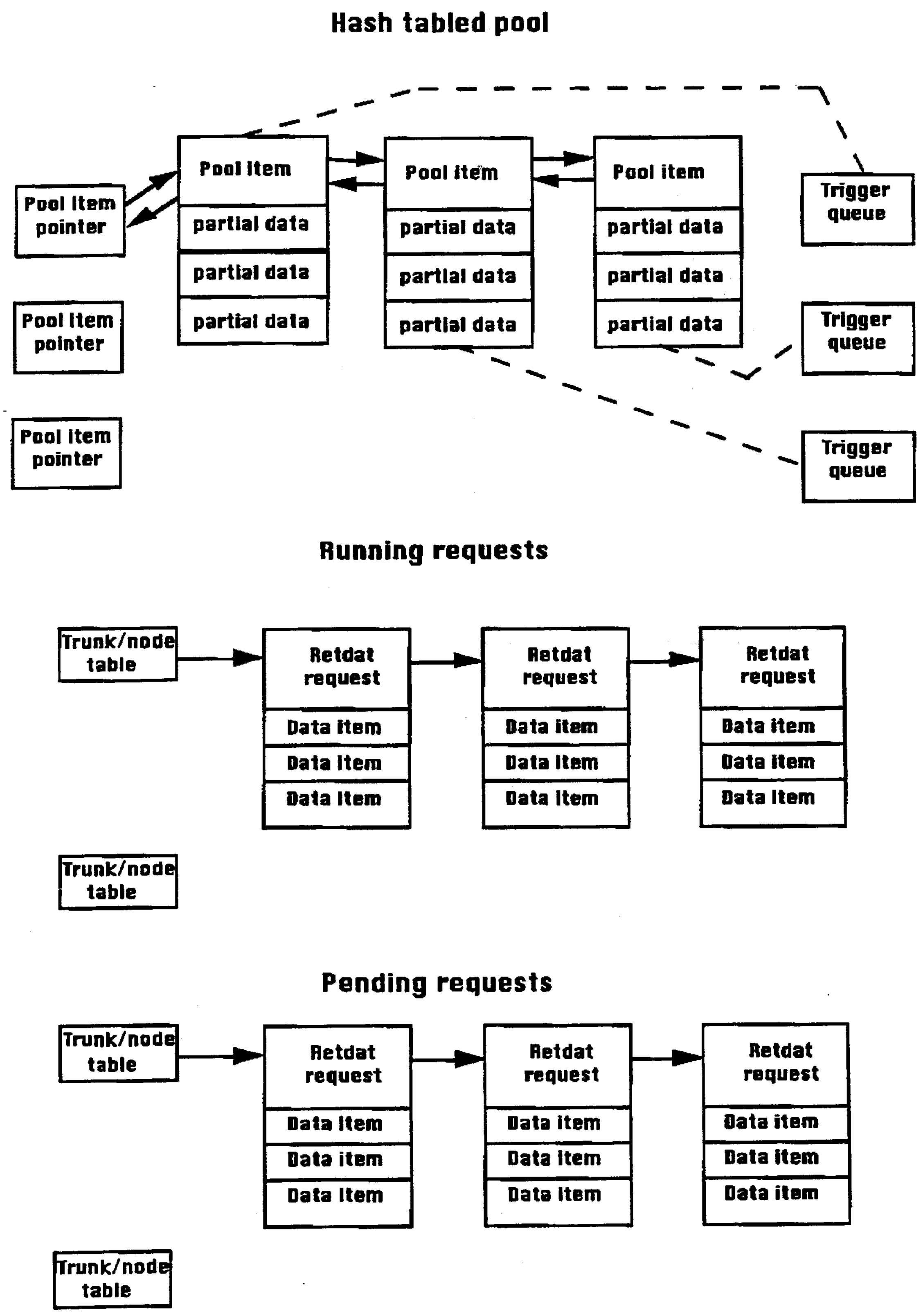


Fig 2. Server/Consolidator main queues

The most complex part of the system is the pool itself. In order to speed up the access every element of the pool is hashed based on the Device Index (DI) and Property Index (PI) that identify uniquely every Fermilab device. Every incoming request is parsed for participating device requests and every device is ether linked with the existing element of the pool or a new pool element is created for this device. Two other queues have to do with the requests built out of original requests. Both of them are organized on the basis of the node number of the system that they are talking to. One of them is the queue of the active (serviced) requests; another one is the queue of pending requests. Usually pending requests are the result of one of the nodes being down. Server/consolidator will periodically try to restart pending requests, so that if the node that was down is rebooted, data will automatically start to come back.

3. Conclusion.

The system is completed and tested. It has been heavily used for cryogenic system support for the last several months.

References

1. M. Glass et al, The upgraded Tevatron Front End. Nucl. Instr. and Meth. a293(1990) 87-90
2. B. Lublinsky, J. Firebaugh, J. Smolucha, New Tevatron Cryogenic Control System, "The proceedings 1993 International Conference on High Energy Physics", V3, p1817.
3. B. Lublinsky. Shell software for Smart subsytems with front-end capabilities. Nucl. Instr. and Meth A 352 (1994) 403 - 406

Multiple alarms, Major Goals and Implementation.

B. Lublinsky.

Fermi National Accelerator Laboratory*

1. Introduction

It is a very common situation that controlled hardware can work in a variety of different regimes, in which case the control system has to be smart enough to recognize a regime change and reconfigure its alarm system accordingly. For example, when a magnet quench occurs, the refrigerators warm up and the cryogenic control system will generate many alarms. Only a few of them are important and they may be lost in the flood of generated alarms. If the alarm system is smart enough to increase permissible maxima for temperatures and pressures, we will see only what is important. Multiple alarms support usually involves three major parts:

- regime determination support;
- regime switching support;
- multiple alarms support.

This paper discuss implementation of multiple alarms support for the Fermilab cryogenic control system (FRIGes) and the ways of possible utilization of the described approach in the rest of our overall control system, long known as ACNET.

2. The overall structure of multiple alarms support.

The regime determination is the less formalized part of multiple alarms support and can range from a brute force setting to a very precise calculation using the system's parameters. The current implementation allows for a wide range of possible approaches by using the FRIG finite state machine mechanism[1]. This mechanism is already built into the cryogenic control software, which is flexible enough that it is relatively easy to implement an additional finite state machine (FSM) which is dedicated solely to tracking of the cryogenic system parameters and determining the current regime.

The regime switching mechanism is significantly more generic and is implemented in the form of a Virtual Machine Front End (VMFE) that can be used not only by the cryogenic control system but throughout the whole of ACNET. The overall interaction scheme for multiple alarms support is shown in figure 1.

Multiple alarm support itself is included in the FRIG microprocessors internally in the form of keeping several alarm blocks for every particular device having this tracking mechanism and also of allowing alarm block switching based on regime changes.

3. Regime determination and broadcasting.

Several changes have been made to support multiple alarms in the FRIG micro. Firstly, two new types of devices were introduced: a virtual machine control device and a virtual machine reflection device (see figure 1). A virtual machine control device in turn consists of two sub-devices - primary and secondary. The secondary device is used for establishing the link to the VMFE. It supports three database properties: setting (setting of the VMFE device to which it will be connected), reading of setting (reading of the set information back) and basic status (acknowledgment that the link to the VMFE has been established).

The primary device is used for setting of the regime itself. The regime can be set either externally via the network from an application or internally by FRIG software, for example by the dedicated FSM. When the primary device receives a setting it sends a corresponding message (via a proprietary protocol) to the VMFE. Its presence facilitates internal intelligence in FRIG micro, which makes the decisions about regime, or state, changes. The database properties supported for the primary device are setting (setting of the current regime), reading and reading the setting (both reading the current regime) and basic status (as for the

* Operated by Universities Research Association under contract with the U.S. Department of Energy

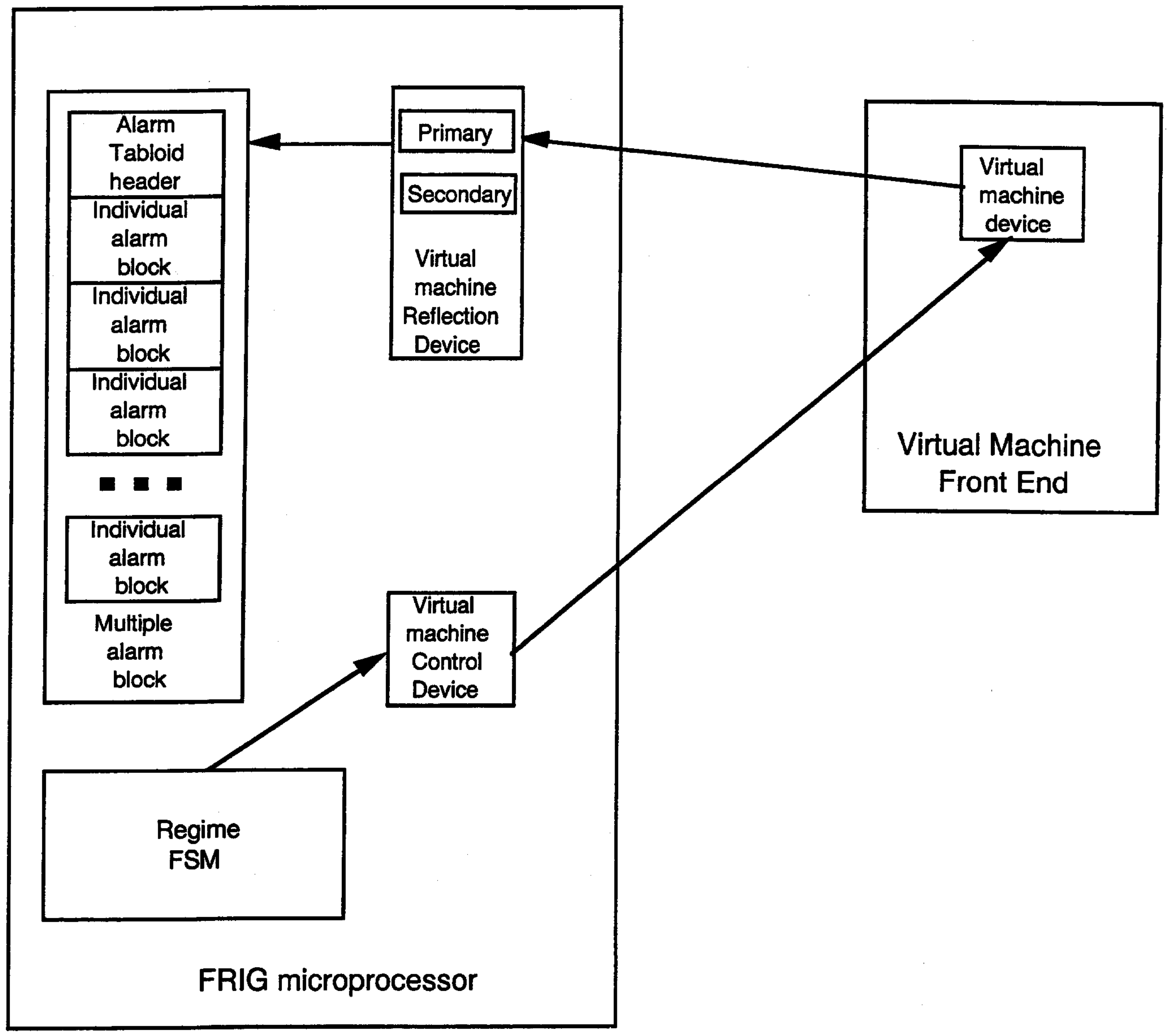


Fig. 1. Overall organization of the multiple alarms support

secondary device). The VMFE, when it receives a regime change message, checks it for correctness and, if it is in fact correct, does the setting to the virtual machine reflection device, which is what actually controls alarm block changing. In the current implementation we are supporting up to 64 virtual machine control devices and up to 64 virtual machine reflection devices. This framework is flexible enough to provide both internal and external regime switching and internode communications.

4. Virtual machine FE.

VMFE is an Open Access Front-end (an ACNET term referring to a node below the consoles in the standard diagram, but where user application code can be inserted) that services requests for state changes of virtual machines. Introduction of this FE might seem as an over-complication of the multiple alarms scheme, but in fact it allows one to solve two very important problems:

- centralization of control of possible states and state transitions for all virtual machines in the control system;
- coordination of regime change between multiple microprocessors.

The first goal is achieved by holding the descriptions of the virtual machines in ACNET devices, owned by the VMFE (usually there is one-to-one correspondence between these devices and virtual machine control devices). Each virtual machine description allows the user to define the set of valid states for this machine and to allow/prohibit state transitions. This significantly simplifies the day-to-day operation of maintaining the current virtual machine definitions.

The second goal is achieved by linking multiple virtual machine reflection devices to one virtual machine device in the VMFE.

5. Microprocessor support for multiple alarms.

Multiple alarm support is achieved by replacement of the usual alarm blocks by alarm tabloids, each containing 16 alarm blocks appropriate for normal devices. Every tabloid has a pointer to the virtual machine reflection device by which it is controlled. This hierarchical scheme is established in the database, by introducing an additional property - PRVMDI (standing for PRoperty Virtual Machine Device Index). Given a VMFE device, it is possible to query it to determine the virtual machine reflection device for the given node, that actually fulfills local regime control. This hierarchy can be downloaded into the microprocessor and queried from a console application, both via a specialized protocol. Each time an alarm scan is performed the value of the virtual machine reflection device is obtained and this value serves as an index into the array of alarm blocks for this particular tabloid. The virtual machine reflection device itself, in this implementation, doesn't have to know about the tabloids it is controlling. This allows one to have connection pointers only in the tabloids themselves and eliminates the necessity of maintaining a linked list of tabloids belonging to each particular virtual machine reflection. The overall regime tracking mechanism is shown in figure 2.

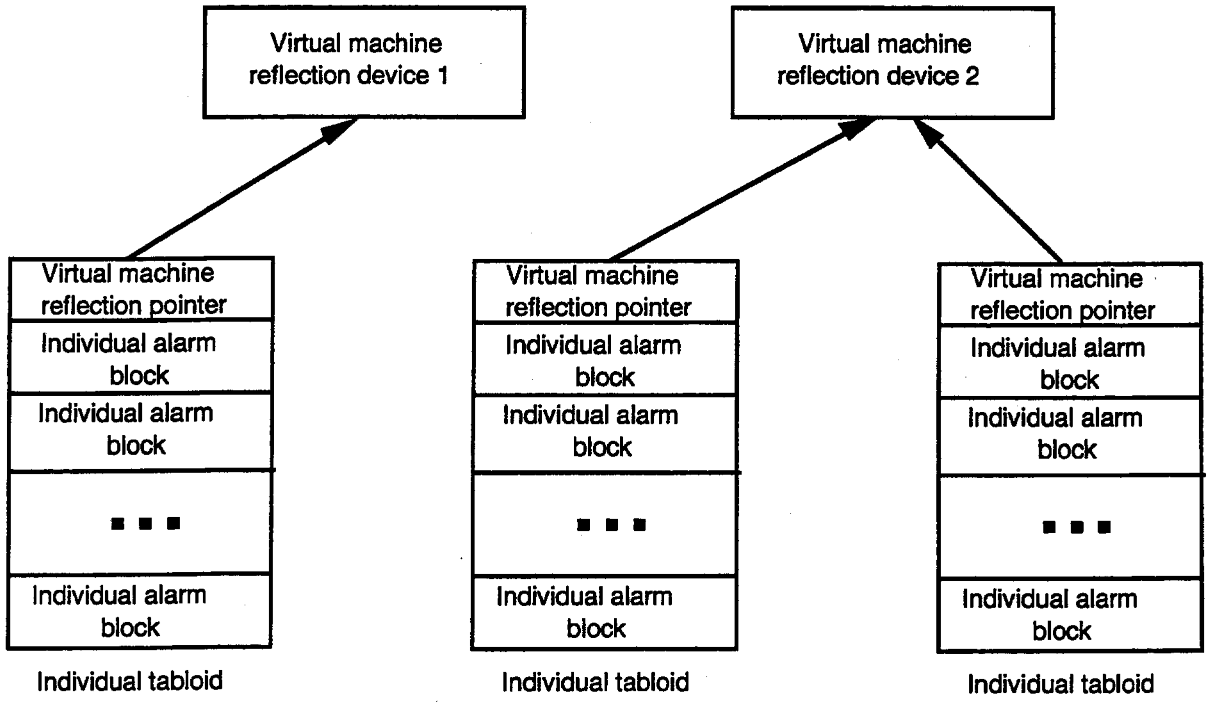


Fig 2. The Tracking mechanism for regime change in alarms support

There are several caveats here. In order not to lose information when a state change occurs, tabloids contain, besides the alarm blocks themselves, the current state value, current GOOD/BAD status, and current "message send to alarm process" flag, thus allowing smooth state transition. Whenever transition occurs the normal alarm processing procedure is followed, beginning with initialization. In the abnormal situations when the virtual machine reflection device requests a state for which the alarm block doesn't exist, the microprocessor will produce an alarm of its own indicating this fact. There is a limited number of such alarms which can be produced, in order to prevent an 'alarm storm' under unusual circumstances.

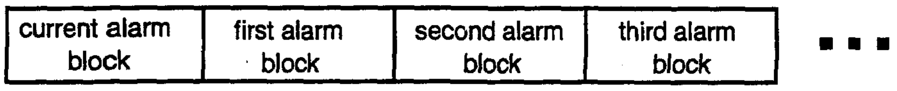


Fig. 3 The external view of the alarm blocks.

The way in which an application sees a multiple alarm block is shown in figure 3. Here "current" stands for the alarm block to which the virtual machine reflection device is pointing. Setting to and reading from this block is possible, independent of what this "current" state is. Similarly the various "physical" blocks may also be referenced.

6. Database support for multiple alarms.

Work had to be done for the ACNET database to support multiple alarms:
- support for the alarm array was implemented;
- the additional property PRVMDI was added.

Support for the alarm array assumes that instead of one alarm block with a particular length, multiple alarm blocks can be associated with every alarm property (analog/digital alarms). The database supports the same "view" of the alarm block tabloid as an external application except that alarm block 0, the "current" block, is not defined in the database.

PRVMDI is that new property introduced especially for multiple alarm support. It is generalized and thus could be used to support 'multiples' of any property. Specifically it consists of two fields; the first of these is the index of the VMFE device, that is making the actual state transition. Using this index an application can query the VMFE about the virtual machine reflection device of interest. The second field is a set of bits corresponding to all possible properties, each of which indicates whether that property will be affected by a VM device transition.

7. Conclusion.

Multiple alarm support is fully operational and has been used in the cryogenic control system at Fermilab for several months. It has proved to be very useful operationally, especially during warm-up and cool-down regimes for the Tevatron. Extension to other ACNET subsystems is under active consideration.

8. Acknowledgments.

Many people participated in design, implementation and testing of multiple alarms. Bob Joshel designed and implemented the Virtual Machine FE. Brian Hendricks, Tim Zingelman, Bob Joshel and Glenn Johnson implemented database support. Alex Martinez of the Cryo Group implemented cryogenic-specific FSM support and thoroughly debugged the whole system.

References

1. B. Lublinsky, J. Firebaugh and J. Smolucha, New Tevatron Cryogenic Control System, "Proceedings of the 1993 Particle Accelerator Conference", V3, p1817.

Development of Device Drivers on Real-time UNIX for the SPring-8 Storage Ring Control System

T. Masuda, S. Fujiwara, T. Fukui, A. Taketani,
R. Tanaka, T. Wada, W. Xu, and A. Yamashita

SPring-8, Kamigori, Ako-gun, Hyogo 678-12, Japan

The HP-RT device drivers and API functions have been developed for VME boards such as DI, DO, TTL DI/O, AI, PTG, GP-IB interface, and RIO (Remote I/O). The drivers access the devices through the SWSM (System Wide Shared Memory) mechanism. Design concepts of the APIs were to provide a simple interface, uniform types of function and arguments and method of handling. The APIs are the only way to access and control the devices from application programs.

1. Introduction

A distributed computing system has been adopted for the SPring-8 storage ring control system [1]. We use several workstations for operator consoles and 28 VMEbus computer systems with HP-RT around the storage ring as the front-end controllers. Each VME system is connected by an optical fiber to a 100Mbps FDDI backbone LAN through a FDDI-Ethernet switching HUB. Hence, each VME system can use the full bandwidth of Ethernet (10Mbps) for communication with upper-level computers. The optical-linked RIO system is mainly used as the field-bus for the control of magnet power supplies, vacuum system, and beam position monitoring system [2].

2. VME system

To consider expandability and maintainability, we have adopted industry-standard hardware as far as possible. Thus we have adopted a VMEbus computer system as the front-end controller. Because of the large number of manufacturers making equipment to this standard, we can take advantage of the various single board computers, I/O boards and other bus/network interface boards available.

The equipment is directly controlled by I/O modules in the VME system or through the RIO or the GPIB field bus as shown in Fig.1.

2.1 CPU board

The CPU board is a HP9000/743rt/64 which has a 32-bit PA-RISC 7100LC with a 64MHz clock. This CPU board is very powerful and performance is quite satisfactory (77.7MIPS) as a front-end controller. It has an AUI interface port, two RS-232C ports, an HP-parallel interface port and a single-ended SCSI II interface on its front panel, and a PCMCIA interface on the mother board. A maximum of 128MB main memory can be fitted on a single-slot board. In our case, 16MB of on-board memory and a 20MB PCMCIA flash ROM card are attached. The flash ROM card is used as a boot device for the operating system (OS).

2.2 I/O boards

We use several types of I/O boards, such as a 64-bit digital input (DI) board, a 64-bit digital output (DO) board, a 96-bit TTL level digital input/output (TTL I/O) board, a 32-channel single-ended/16-channel differential analog input (AI) board and a 5-axis control pulse-train generator (PTG) board, mainly for RF equipment control.

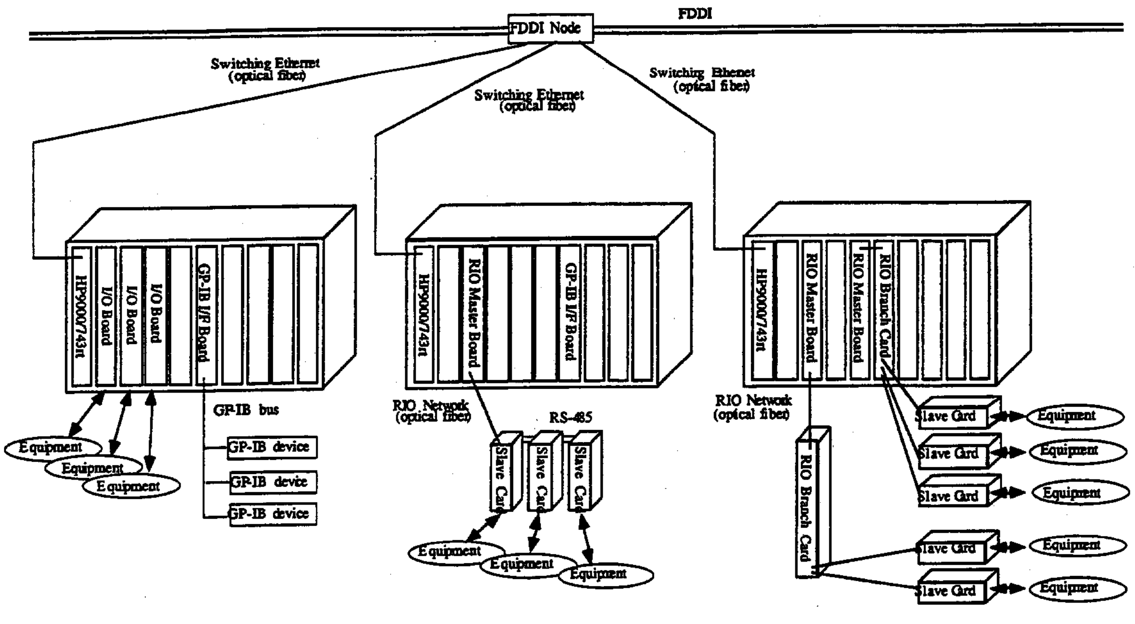


Fig.1. Schematic diagram of VME system and field buses.

2.3 Field buses

A fiber distributed RIO system was originally developed for the control of the magnet power supplies. The purpose of the development was electric isolation, noise rejection and economy.

The RIO system consists of the master boards, several types of slave cards and 8-port branch cards if needed. Optical fiber cables are used for connection in a star topology between the master board and the slave cards. The master is a VME board with a 4KB dual port RAM. Currently, six types of slave cards are available. These are Euro cards but not VME boards. The specifications of each slave card are as follows:
- Type-A: 1-channel AI with 16-bit resolution (integration type, conversion time < 128msec), 1-channel analog output (AO) with 16-bit resolution, 8-bit DI and 8-bit DO,
- Type-B: 32-bit DI and 32-bit DO,
- Type-D: 40-bit DI,
- Type-E: 1-channel AI with 16-bit resolution (integration type, conversion time < 32msec), 8-bit DI and 8-bit DO,
- Type-F: 4-channel AI with 12-bit resolution (successively conversion type, conversion time < 1msec) with 32KB local buffer and 1-bit external synchronization signal input,
- Type-G: 16-channel DI and 64-bit DO.

The features of the RIO system are as follows:
- optical fiber star connection between master board and slave cards,
- serial-link communication,
- maximum of 62 slave cards can be connected per one master board,
- 1Mbps transmission rate,
- cyclic transmission needs 0.2msec for type-A, B, D, E card, 0.5msec for type-G card and 27msec for type-F card for communication,
- up to 1 km transmission distance,
- HDLC protocol,
- can be connected by twisted pair cable (RS-485).

GPIB is also used as the field bus and largly used for the control of intelligent devices such as DVMs.

3. Operating system

3.1 Selection criteria

The selection criteria of the OS for the front-end controller were mainly real-time features and open system features, and that it should be compliant with POSIX which is a standard OS proposed by IEEE. Among such OSs, we chose LynxOS based HP-RT. The LynxOS supports many platforms such as i80386/80486/Pentium, M68030/40, SPARC2/microSPARC, R3000 and PowerPC 601/603/604. The HP-RT is a migrated OS of LynxOS to PA-RISC. Availability is restricted because HP-RT was originally only running on particular CPU boards such as HP9000/742rt or 743rt. However we expect better support from the company. It is a crucial point in reducing development cost and time for the new OS.

3.2 Features of HP-RT

HP-RT is just UNIX with real-time features, so it is not difficult to understand and manage. Actually we can get the same development environment and use the same commands as UNIX. LynxOS provides both self- and cross-development environments. On the other hand HP-RT provides only a cross-development environment and a host system is required. HP9000/700 or 800 series workstations are available as the host computer of HP-RT. Application programs and the kernels for the target system are built on the host system. HP-RT provides two types of kernel, one is RAM-based and the other is disk-based. In the case of RAM-based kernel, it is booted from the host system through the network with BOOTP (BOOTstrap Protocol), so the host must be the BOOTP server for many target systems with different kernels. At the same time, the host system has to provide the building environment for each kernel which has a different configuration. In other words, the host system has to provide both the download service for the specified kernel and the development environment for a different kernel. When the first installation of the specified kernel has been madeto the local mass storage device and problems arise, we must consider the same problem for the disk-based kernel This setup is a little complex. From the point of view of the management of the target system, the host-target system is more preferable.

4. Device driver

4.1 Feature of the HP-RT device driver

Device drivers are embedded in the OS kernel and are the glue between the kernel and the devices in the VME system. They convert the commands from the kernel to the devices and vice versa. By separating the drivers from the kernel itself, hardware independence of the original kernel is kept.
The HP-RT device driver is a collection of the functions called "entry points". The kernel can access the driver only through these entry points. The HP-RT device driver has nine entry points as shown in Table 1.

Table 1
Name of entry points of HP-RT device driver and its purpose

Entry Point	Purpose
install	Called to install a major device
uninstall	Called to uninstall a major device
open	Called when device is opened
close	Called when device is closed
read	Called to read data
write	Called to write data
ioctl	Called to control a device
select	support select system call
strategy	called to perform block read and write operation (only for block device)

The HP-RT device drivers can be written in C. Many special library functions with a C interface are provided because almost all ordinary C library functions are not available in the drivers. They are called "driver service calls". They provide the functions such as printf for kernel debugging, dynamic memory allocation, hardware interrupt configuration, kernel thread management, 10msec and 1ms resolution timer interrupt settings, system semaphore control, address conversion and so on. The driver accesses the devices in the VME system through the SWSM (System Wide Shared Memory) mechanism by which any VME address within the specified VME space is mapped into the virtual address of the kernel. By means of the SWSM mechanism and some particular service calls, we don't have to take care of the mapping of the virtual address corresponding to the VME address.

Three types of hardware interrupt handler are supported, SIGNAL_ONLY, ASM_CALL and C_CALL. The handler of ASM_CALL (written in PA-RISC assembly language) and C_CALL (written in C) are called by the interrupt dispatcher directly, while the handler of SIGNAL_ONLY is the kernel thread to wait for the system semaphore from the dispatcher.

4.2 Design concepts

Design concepts of our device driver are generality, maintainability and reliability. We should realize the primitive functions and eliminate any sequence depending on an application program, because the driver has to be modified when this sequence is changed. Such particular sequences should be realized as libraries outside of the drivers.

With regard to RIO, we should not implement the control sequences of each slave card into the device driver. When an application programmer wants to control the slave card directly, he does not want to know how to control the master. We realize such particular functions by creating libraries. These libraries are a part of the device drivers in a wide sense.

4.3 Status

The early versions of the device drivers for I/O boards have been developed except for the TTL DI/O, the RIO master board and the GPIB interface board. Hardware interrupts are supported in the AI, DI, PTG and GPIB interface boards by the SIGNAL_ONLY type.

We start the test and the modification of the RIO device driver because the test of our control software scheme, especially the test of the *EM (Equipment Manager)*, is planned to handle the steering magnet power supplies [3]. They are controlled through the RIO and interfaced to type-A slave cards.

Through the test of APIs of the RIO(see 5.2), the RIO device driver was confirmed to work well. This RIO device driver was installed in the real control system which has 7 RIO master boards. We checked that all master boards are controlled through the driver.

5. API for device handling

5.1 Design concepts

We have designed the application program interfaces (APIs) for the application programs such as *EM* to handle the devices in the VME system. Any application program should be written by using the APIs and never by touching the drivers directly by using system calls or libraries.

The design concepts of APIs are the following:
- provide flat and primitive APIs,
- provide uniform type of function and arguments and manner of handling.

5.2 Status

The first version of the APIs for open/close the device, RIO master board control and type-A and type-B slave card control have been developed. The master board is identified by the file descriptor which is a return value from the open function of the device. The slave card is identified by the slave number. Thirteen APIs have been developed for master control which involve the initialization of the master, control of the transmission, execution of the self-check and getting the status. Additionally 6 APIs for type-A slave and 3 APIs for type-B slave are provided for data taking and reading.

The primary system was found to work well. Actually these APIs were linked with the *EM* running on HP-RT for the control of the steering magnet. Two steering magnet power supplies were controlled. We could successfully set the power supplies to on/off, reset the error status, set and read current value and read the status with the RIO APIs and the *EM*.

References

[1] R.Tanaka et al., these proceedings (ICALEPCS'95, Chicago, USA, 1995)
[2] H.Takebe et al., Proc. of the 4th European Particle Accelerator Conf., London, June 1994, Page 1827-1829
[3] A.Taketani et al., these proceedings (ICALEPCS'95, Chicago, USA, 1995)

Automatic High Voltage Conditioning of the Electrostatic LEP Separators without Conventional Programming

B. Balhan, A. Burton, E. Carlier, J. H. Dieperink and V. Mertens.
European Organization for Nuclear Research (CERN) - SL Division,
CH - 1211 Geneva 23, Switzerland

Abstract

The TS Tool Kit is a generic, fully data-driven and user-configurable software system developed at CERN for supervisory, control and data acquisition applications. It provides a comprehensive framework to solve fairly complex process control problems requiring response times of the order of a second, without any need for conventional programming. The characteristics and benefits of this approach are discussed in the context of the new high voltage conditioning process for the electrostatic LEP separators.

1. EQUIPMENT

Separation of the e+e- beams in LEP is achieved by a vertical electrostatic separation system comprising 40 bipolar separator units, arranged to provide local vertical separation bumps at all eight LEP interaction points. Each separator consists of a vacuum tank of 50 cm diameter and 4,5 m length and contains two high-voltage electrodes mounted on two insulating supports as shown in figure 1. Each electrode is connected via an alumina feedthrough to a high-voltage (HV) generator (± 175 kV DC). The gap width can be varied between 6 and 16 cm [1].

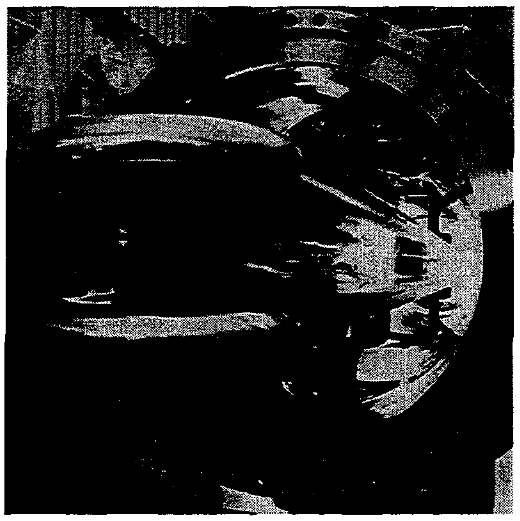


Fig 1. Separator tank with electrodes.

In order to improve the HV behaviour at the nominal field (15 - 30 kV/cm) after assembly, loss of vacuum, or polarity change, a separator is (re-)conditioned by increasing the voltage slowly to the maximum field (56 kV/cm). The maximum speed of voltage increase is determined by two parameters: the spark rate and the vacuum pressure in the tank. After the maximum field has been reached and maintained during several hours for consolidation, the field is reduced to 50 kV/cm for a total of 72 hours, during which the tank is checked for a low spark rate.

2. PROJECT OVERVIEW

To speed up the procedure and to save manpower a software regulation process has been used for many years to perform the HV conditioning automatically. Its original implementation was a stand-alone C program [2] designed to run on the local process controllers (PCs under SCO XENIX) widely used for LEP control. For screen handling, user input, and display of historical and trending curves the program made abundant use of the CGI (Computer Graphics Interface) graphics package, conveniently available under XENIX. Many of the parameters needed during the conditioning, like maximum voltages, flat-top timeout interval, etc., were hard-coded, as the program was only foreseen to be used with the initial separator setup [1]. As it was conceived, this program could also run only locally and not be interrogated or influenced over the network. Therefore, its use necessitated a lot of travel between the equipment specialists' offices and the various LEP interaction points.

The recent migration from XENIX to LynxOS [3] (which no longer offers CGI) as operating system in the process controllers, the evolution of the separator layout over the years (e.g. for the 'LEP bunch train scheme' [4] or LEP2) has rendered this old implementation of the automatic conditioning obsolete. Therefore a new solution has been recently conceived and realised.

3. CHARACTERISTICS

The new automatic conditioning process has been developed in the framework of the fully configurable, event and data driven 'TS Tool Kit' described already elsewhere [5, 6]. This provided:

- No need for conventional (e.g. C) programming;
- Portability (runs so far on HP-UX, ULTRIX, and LynxOS) (applications are platform-independent);
- Full on-line remote control;
- Separation between algorithm and user interface;
- Equipment independence (can be used for all types of LEP separators but also other equipment, e.g. the SPS extraction electrostatic septa).

In particular the suppression of conventional programming has allowed one to set up the new system in a remarkably short time, permitting concentration on the proper implementation of the conditioning algorithm. It largely favoured a clear communication between users and developers, resulting in a successful transfer of the desired functionality into the final product. The user interface was largely conceived by non-software specialists, using functions of the TS Tool Kit together with other easily configurable tools [7]. Non-software specialists will also be able to perform most of the future upgrade work in case the applications evolve.

4. TS TOOL KIT

The core of the TS Tool Kit [5, 6] is a shared database to which the access is handled by a client/server mechanism. This database is the run-time repository for all the data needed to run the application. It is surrounded with a set of client modules, each dealing with a specific control task, like data acquisition, running control algorithms, or user interaction. The general application behaviour is given through flat data files which are read in and interpreted at startup. The TS Tool Kit configuration needed to implement the automatic conditioning process is shown in figure 2.

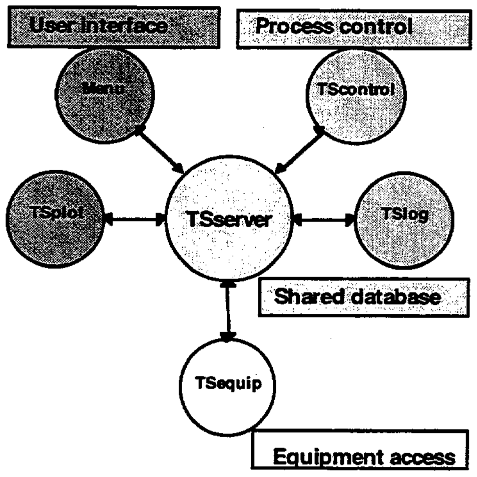


Fig 2. Software layout.

The various modules perform the following functions:

TSserver:	shared database.
TScontrol:	runs control and supervision algorithms specified in 'rules'.
TSlog:	stores, at definable time intervals or on change, any set of data in the shared database onto files or onto a remote Oracle database.
TSequip:	sends commands to, and reads data from, the equipment.
TSplot:	writes selected data onto files in a format understandable by 'gnuplot' to provide trending curves.
Menu:	generic alphanumeric user interface which permits, based on simple text files, the organization of operating system commands or calls to user-written programs into a tree of menus, to display the menus, to move in the menu tree and to invoke commands by single-key actions [7].

Several auxiliary modules (not shown here) are used from within Menu, for user access to the shared database for information and run-time control of the application.

5. SCENARIO and ALGORITHM

The automatic conditioning procedure is determined through 'scenarios', made up from a set of parameters and 'rules', executed by the TScontrol module. Various scenarios are available for positive, negative, or bipolar conditioning, selectable from Menu. Likewise, parameters are pre-set as appropriate for the chosen scenario, but can be modified dynamically at run-time to influence the procedure.

The rules are formulated as conditions and subsequent actions in term of data in the shared database, and specified in a simple "controls language" [8]. These rules are read from a text file at start-up and continuously evaluated at run-time. All rules are executed completely asynchronously when their respective conditions (WHEN ...) are evaluated as 'TRUE'.

A largely simplified version of the rules constituting the automatic conditioning algorithm is shown in figure 3.

The overall process regulation loop is made up of three parts: hardware acquisition, algorithm checking, and hardware setting modification. A frequency of 1 Hz has been chosen for the acquisition providing a sufficiently fast reaction time in case of vacuum degradation or sparks (spark bursts are interlocked by software in the HV generators itself). It is noteworthy that the TS Tool Kit is able to provide considerable shorter reaction times if run on sufficiently performant hosts. Separator conditioning for the whole of LEP is presently run from a central HP workstation, even when several tanks need to be conditioned in parallel. Due to the full platform independence of TS Tool Kit applications, individual processes could be also be run locally or separately for each interaction point, without any modification.

```
WHEN Maximum electrical field reached
   DO
      IF First time
         THEN
            Double charging current
         ELSE
            Check spark rate
      ENDIF
   ENDDO
WHEN Poor vacuum
   DO
      Decrement charging current
   ENDDO
WHEN Sparks
   DO
      Decrement charging current
   ENDDO
WHEN Conditioning rate too low
   DO
      Increment charging current
   ENDDO
WHEN Conditioning rate too high
   DO
      Decrement charging current
ENDDO
```

Fig 3. Set of TScontrol rules constituting the main conditioning algorithm (largely simplified).

In terms of numbers the new algorithm comprises 25 rules with a total of 500 lines. In comparison, the previous implementation comprised about 6500 lines of C code and some minor parts in assembler.

6. CONCLUSION

Despite its considerable advantages in flexibility and user-friendliness over the previous solution, the new automatic conditioning of the LEP separators has been realised, using the TS Tool Kit software package, in a much shorter time scale (ca. 2 man-months counted from after the user specification phase compared to ca. 1.5 man-years for the old process). All tests have so far been successfully achieved. The process will be intensively used during the coming LEP shutdown. The new solution will be much easier to maintain and adapt to new needs if necessary, even outside its original LEP applications.

As the TS Tool Kit is nicely integrated into the CERN SL accelerator control system (offering interfaces to the X user interface, the general CERN alarm system, and Oracle on-line databases) further extensions of the automatic conditioning, like alarm generation and forwarding or graphical user interfaces are quite straightforward.

ACKNOWLEDGEMENTS

The authors would like to thank Noel Garrel for providing the initial user specifications of the automatic conditioning process, Brennan Goddard for his help and suggestions, and Alberto Aimar for upgrading the TScontrol module.

REFERENCES

[1] W. Kalbreier, N. Garrel, R. Guinand, R. L. Keizer, K. H. Kissler, Layout, Design and Construction of the Electrostatic Separation System of the LEP e+e- Collider, Proc. European Accelerator Conference, Rome, Italy, June 7-11, 1988, Vol. 2, pp 1184 - 1186.

[2] B. James and L. de Lavallade, private communications.

[3] A. K. Brignet, A. Burton, E. Carlier, J. H. Dieperink, B. Goddard, M. Laffin, M. Lamont, V. Mertens and H. Verhagen, The Evolution of LEP Separator Controls in the Wake of the Bunch Train Project, CERN SL/Note 95-47 (BT).

[4] B. Balhan, A. Burton, E. Carlier, J.-P. Deluen, J. H. Dieperink, N. Garrel, B. Goddard, R. Guinand, W. Kalbreier, M. Laffin, M. Lamont, V. Mertens, J. Poole and H. Verhagen, Modification of the LEP Electrostatic Separators Systems for Operation with Bunch Trains, Proc. 1995 Particle Accelerator Conference and International Conference on High-Energy Accelerators, Dallas, USA, May 1 - 5, 1995, CERN SL/95-45 (BT).

[5] V. Mertens, A. Aimar and E. Carlier, A Simple Generic Software Tool Kit for Distributed Controls Applications, Proc. International Conference on Accelerators and Large Experimental Physics Control Systems, Berlin, Germany, October 18 - 22, 1993, Nucl. Inst. Meth. A352(1994)427, CERN SL/93-49 (BT), and World Wide Web URL http://dxbt00.cern.ch/ts/ts_papers/berlin93.html.

[6] A. Aimar, E. Carlier and V. Mertens, An User- Friendly Approach to Process Control Software in the Framework of a Generic Tool Kit for Distributed Application, Workshop on Workstation & Software Tools for Automatic Control, Prague, Czech Republic, October 26 - 27, 1993, CERN SL/93-50 (BT) and World-Wide Web URL http://dxbt00.cern.ch/ts/ts_papers/prague.html.

[7] V. Mertens, Menu - A Multi-Platform Alphanumeric User Interface - Version 2 User Guide, CERN SL/Note 95-107 (BT) and World-Wide Web URL http://dxbt00.cern.ch/menu.html.

[8] A. Aimar, E. Carlier, C. Cameron, A. Ferrari, B. Goddard, M. Laffin, V. Mertens and M. Tyrrell, Centralised Equipment Surveillance Using a Configurable Software Tool Kit as Front-End to CERN Alarm System, CERN SL/95-34 (BT).

DESIGN OF THE BEAM ORBIT FEEDBACK SYSTEM FOR THE TRISTAN LIGHT SOURCE STUDY

T. Mimashi, A. Akiyama, H. Fukuma, M. Tobiyama
KEK, National Laboratory for High Energy Physics
1-1 Oho, Tsukuba-shi, Ibaraki-ken 305, Japan

S.Yoshida
Kanto Information Service
8-21 Bunkyo, Tsuchiura-shi, Ibaraki-ken 305, Japan

ABSTRACT

A beam orbit feedback system was designed and built for the TRISTAN Light-Source Study. Some of the experiments which use TRISTAN Main Ring as a synchrotron-light source require very high beam position and slope stability. The beam orbit feedback system eliminates oscillations of the beam in time at the center of an undulator. The feedback system is composed of 2 single-path beam monitors and 4 each of horizontal and vertical steering magnets. The control system was built with EPICS.

INTRODUCTION

TRISTAN, the electron-positron colliding ring for high energy physics, has been in operation since 1986. The experiments with this facility were completed in June 1995 and the TRISTAN Main Ring was altered to be utilized as an extremely intense and highly advanced light source facility. An undulator was installed for purposes of study, with experiments planned from September to December 1995 [1][2]. In order to provide high quality light to the experiments the accelerator must satisfy tight tolerances and have low emittance and high current. The beam orbit feedback system was designed to satisfy one of these requirements, the stability of the beam position and slope at the light source point.

THE REQUIREMENTS FOR THE FEEDBACK SYSTEM

Since our beam line is long (about 100 m), the requirements for stability of beam position and slope are relatively tight. *Figure 1 shows them for several experiments.*

	Experiments	Position Stability	Slope Stability
Horizontal Direction	All	± 1500 (μm)	± 15 (μrad)
Vertical Direction	Muscle fibers	± 500 (μm)	±10 (μrad)
	The microbeam	± 50 (μm)	± 10 (μrad)
	Solid wave	± 100 (μm)	± 5 (μrad)
	X-ray parametric scattering	± 50 (μm)	±10 (μrad)
	Nuclear resonance scattering	± 500 (μm)	± 5 (μrad)

Fig. 1 Requirements from each experiment

In order to design the feedback system, we first measured relatively slow closed orbit distortions (timescale from 10 minutes to 8 hours) and fast beam vibrations (from 3 to 100 Hz). From the closed orbit distortion measurement, assuming that the orbit distortion was caused by the change of a quadruple magnet position, we can estimate that magnets move 1.7/2.6 μm (at the one sigma level) in the horizontal/vertical direction over 8 hours. This corresponds to the light source position moving 48/94 μm (r.m.s.) in the horizontal/vertical direction and the slope moving 14/21 μrad (r.m.s.). The fast beam vibration (3 to 100 Hz) was also measured. The beam position vibrates 0.22/0.40 μm in the horizontal/vertical direction at 5 Hz and the corresponding slope changes 0.05/0.08 μrad. From these measurements we thus conclude that we can ignore the fast beam vibration.

FEEDBACK DESIGN

Figure 2 shows a schematic of the feedback system. For H, to approximate both the response of the steering magnets and an eddy current effect of the vacuum chamber, we write the transfer function as:

$$\frac{1}{\frac{1}{85}s+1} = \frac{85}{s+85}$$

where 1/85 comes from the measured field of the steering magnets and s is the usual imaginary angular frequency ($s=i\omega$). The Z-transform, appropriate for discrete time sampling, is:

$$H(z) = \frac{1-e^{-aT}}{z}(z-e^{-aT})$$

where $a=85$ and T is the sample time. Similarly the Z-transform of the control function becomes:

$$G(z) = \frac{K}{z}\left(1+\frac{T}{T_I\left(1-\frac{1}{z}\right)}+\frac{T_D\left(1-\frac{1}{z}\right)}{T}\right)$$

where K, T_I and T_D are constants. The three terms inside the large parentheses are the familiar proportional, integral and differential contributions. Finally the closed loop gain as a function of disturbance frequency (f) can be written as:

$$g(f) = \frac{\text{error(e)}}{\text{disturbance (d)}} = \left|\frac{1}{1+H(z)+G(z)}\right| .$$

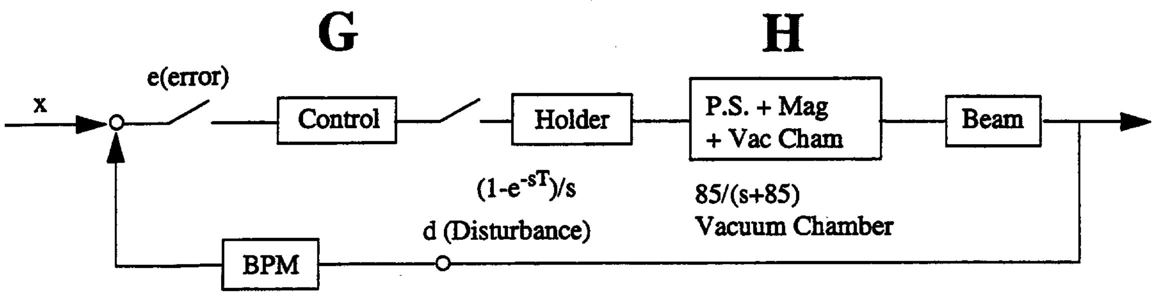


Fig. 2 Schematic view of the feedback loop

HARDWARE IMPLEMENTATION

Figure 3 shows a rough sketch of the beam line around the undulator. The STFH and STFV are the horizontal and vertical steering magnets of the feedback system and the STH and STV are normal horizontal and vertical steering magnets. The beam position and slope at the center of the undulator are calculated from the two recently implemented beam position monitors that are installed on both sides of the undulator and are adjusted with 4 each of vertical and horizontal steering magnets.

Steering magnets

In order to get fast response, laminated iron-core magnets were used for steering. They were designed to control the beam position to 1500 μm in the horizontal direction and 5 μm in the vertical, which correspond to kick angle accuracies of 150 μrad horizontal and 5 μrad vertical. In order to get these slope accuracies at the center of undulator, the steering magnet kick angles must be good to 7.5 μrad/2.5 μrad.

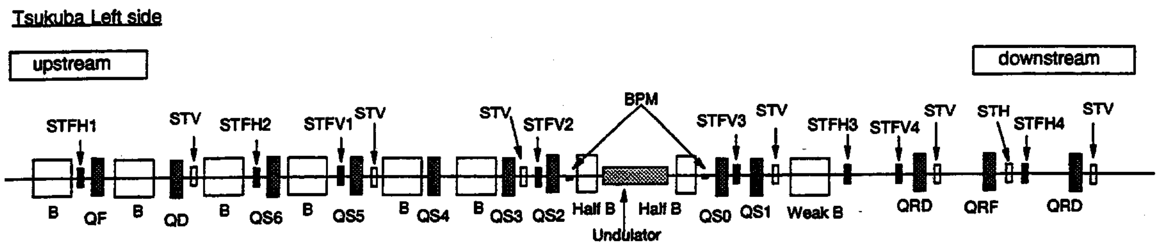


Fig. 3 Schematic view of the beam line around the undulator

Beam position monitors

Two new monitors can measure turn by turn beam positions. Figure 4 shows a block diagram of the read-out system, based on the AM/PM method [4], with 10-bit digitizations. In order to measure the beam position with an accuracy of 50 μm, 100 turn data are accumulated in the memory and used to calculate the beam position.

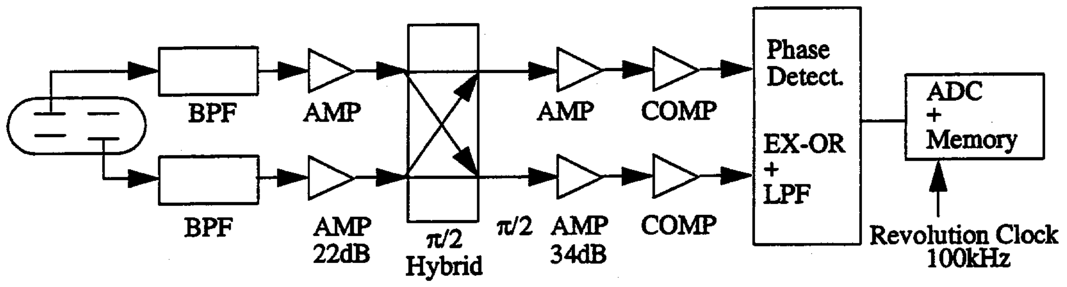


Fig. 4 Block Diagram of the AM/PM method beam monitor readout system

CONTROL SYSTEM

The control system was built based on EPICS; Figure 5 is a block diagram. One VME board (Force 25MHz clock CPU40) and an HP9000/755 were used as control computers. To build the control system some EPICS tools were used. The graphic user interface was written in MEDM, and beam position/slope data were collected by EPICS archive tools. Although EPICS sequencer supports data update of up to 60 Hz, we extended this to 200 Hz by using the auxiliary clock of the VME CPU board.

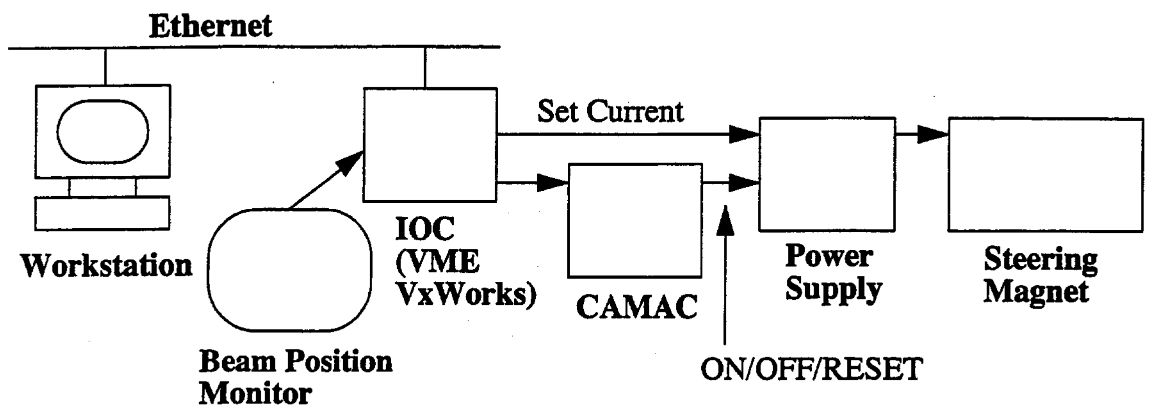


Fig. 5 Block diagram of the control system

FEEDBACK SYSTEM PERFORMANCE

Figure 6 shows a preliminary result of the beam orbit feedback system performance. The feedback frequency was set to 100 Hz and the beam position and slope were calculated with 100 turn data. Figures 6 (a) and (b) show vertical beam positions when the feedback was off/on. Figures 6 (c) and (d) show the corresponding slopes. When the feedback was turned on the beam positions could be made stable within ± 30 μm in both directions, and similarly the slope within 5 μrad.

CONCLUSION

The beam orbit feedback system was built to make the position and slope of the beam stable at the center of an undulator. The control system was designed based on EPICS. When feedback was turned on the requirements for beam stability were satisfied.

REFERENCES

[1] "The Tristan Super Light Facility Conceptual Design Report 1992", KEK Progress Report 92-1

[2] S. Kamada et al., "Accelerator plan for a light-source study at the TRISTAN MR", Rev. Sci. Instrum, 66 (1995) p. 1913

[3] L.R. Dalesio et al.,'EPICS Architecture", ICALEPCS 91, KEK Proceedings 92-15 (1992) p. 278

[4] S.P. Jachim et al., "An RF Beam Position Measurement Module for the Fermilab Energy Doubler", IEEE NS-28 (1981) p. 2323

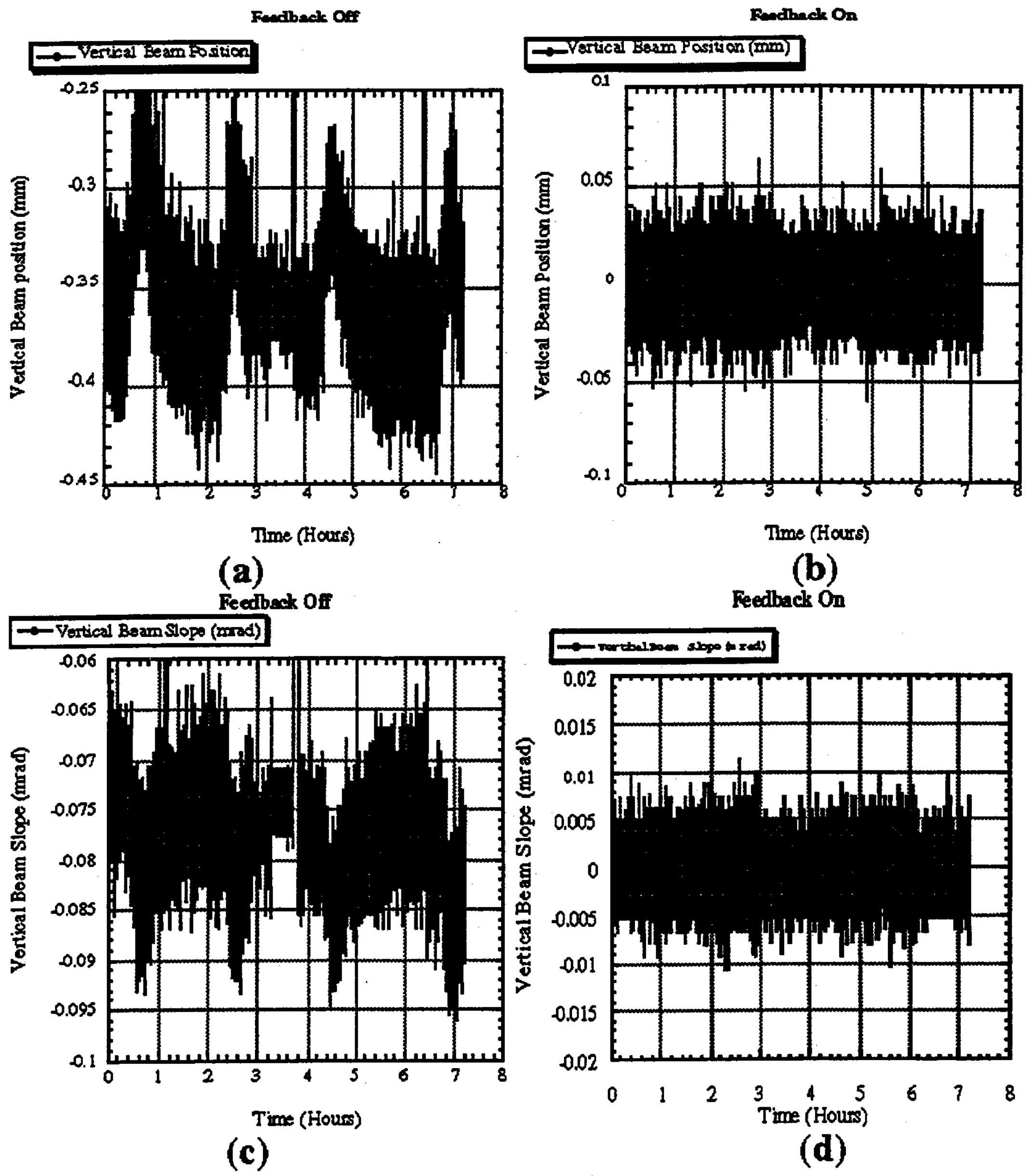


Fig. 6 Vertical beam position and slope at the center of undulator

Monitoring Commercial Conventional Facilities Control with the APS Control System - The Metasys-to-EPICS Interface

G.J. Nawrocki, C. L. Seaver, J.B. Kowalkowski
Advanced Photon Source
Argonne National Laboratory

ABSTRACT

As controls needs at the Advanced Photon Source matured from an installation phase to an operational phase, the need to monitor the existing conventional facilities control system with the EPICS-based accelerator control system was realized. This existing conventional facilities control network is based on a proprietary system from Johnson Controls called Metasys. Initially read-only monitoring of the Metasys parameters will be provided; however, the ability for possible future expansion to full control is available. This paper describes a method of using commercially available hardware and existing EPICS software as a bridge between the Metasys and EPICS control systems.

THE APS CONVENTIONAL FACILITIES CONTROL SYSTEM

The contractor selected for conventional facilities control was Johnson Controls, an environmental control system contractor familiar to many institutional and industrial facilities. This control system, called Metasys, is installed throughout the APS site.

Metasys is a distributed system of intelligent I/O processors, called network control modules (NCM), for control and monitoring of environmental and HVAC parameters. The data link and physical layer of this distributed system uses ARCNET for I/O processor intercommunication as well as the link to operator workstations for the human interface. This ARCNET-based communication path is called the N1 LAN [1]. As with many industrial control systems, the communication protocol and software for this system are proprietary to the manufacturer.

It should be noted that the NCM modules themselves have a subnet for communication to smaller distributed I/O control processors. This subnet, called the N2 Bus, is for communication between I/O processors only and does not provide an interface to an operator workstation [1].

The software native to the Metasys control system is called the Metasys Facilities Management System (FMS). The FMS is configured via the operator workstation, a PC-compatible computer running the Metasys Person Machine Interface (PMI) software. Physical control parameters, such as an individual temperature sensor or the on/off status of a certain water pump, are mapped to analog or binary Metasys objects. Associated with the Metasys objects are parameters which govern such elements as units (analog objects have single unit parameters, binary objects have two unit parameters, one each for the logical "1" and logical "0" values), display precision (analog only), object status (binary only), and of course the present value of the object [2].

A collection of Metasys objects is referred to as a Metasys system. The entire APS conventional facilities control system can be viewed as a tree of Metasys system directories containing files representing Metasys objects. The current software release of the Metasys File Management System at the APS allows for a maximum of 50 Metasys objects per Metasys system.

THE APS ACCELERATOR CONTROL SYSTEM

The accelerator control system at the APS is a distributed, networked system based upon the standards of EPICS [3]. It provides elaborate control and monitoring of all aspects of accelerator operation. Applications written specifically for EPICS provide facilities for data acquisition, back up and restoration of accelerator parameters, as well as the handling of alarms generated by the control system.

EPICS provides for VME-based input/output controllers (IOCs) to be distributed throughout the facility and interconnected via Ethernet to one another and also to UNIX-based operator interface (OPI) consoles [3]. This control system operates and monitors all functions of accelerator systems such as magnet power supplies, vacuum pumps, and beam position monitors.

Physical I/O points in EPICS are mapped to "process variables." Each of these process variables has parameters associated with it which define display and alarm parameters as well as the actual value of the process variable. These parameters are called fields. A collection of process variables in EPICS is referred to as a "database." The database is loaded into the IOC associated with the physical I/O point hardware. Process variable fields are available to other IOCs and OPIs via a communication protocol called channel access [3].

It is clear that the initial step in interfacing the Johnson Controls Metasys System to EPICS is to provide the requisite hardware and software links so that the Metasys object parameters could be mapped to EPICS process variable fields.

THE METASYS NETWORK PORT

The Metasys network port is a device which allows monitoring and controlling of the Metasys Facilities Management System from a third-party computer system. The network port emulates an Allen-Bradley PLC-5 and will exchange FMS data with a system that communicates with Allen-Bradley PLC-5 Word Range Read, Word Range Write, and Read-Modify-Write commands [2].

The network port must be actively programmed to map Metasys objects to the Allen-Bradley PLC-5 syntax. Up to 800 analog Metasys objects may be assigned to the PLC-5 syntax F8:0 - F8:799. Only the present value parameter of an analog object is available in four-byte analog words. Up to 3200 binary Metasys objects may be assigned to the PLC-5 syntax B3:800 - B3:3999. Binary Metasys object information is available in two-byte binary words. Individual bits of this binary word can be parsed to read such binary object parameters as present value and the status parameters: on-line / off-line, defined / undefined, and reliable / unreliable status. Status parameters for analog objects are available for F8:0 - F8:799 from B3:4000 - B3:4799, respectively. The present values of these binary objects are meaningless [2].

The active programming of the network port is accomplished with the Metasys Person Machine Interface (PMI) software. The process involves copying the existing Metasys objects one wishes to monitor from its location in the FMS directory structure of Metasys systems into a directory structure of Metasys systems native to the network port. This can become quite tedious for a large number of Metasys objects and is bound by the finite number of Metasys objects the network port can translate to PLC-5 syntax.

Physically the network port communicates with the Metasys network via ARCNET and appears much like a network control unit as a N1 LAN node. Communication to the PLC-5 system is accomplished via RS232 using the Allen-Bradley Full-Duplex Data Link Layer Protocol (DF/1) at a data rate of 9600 baud.

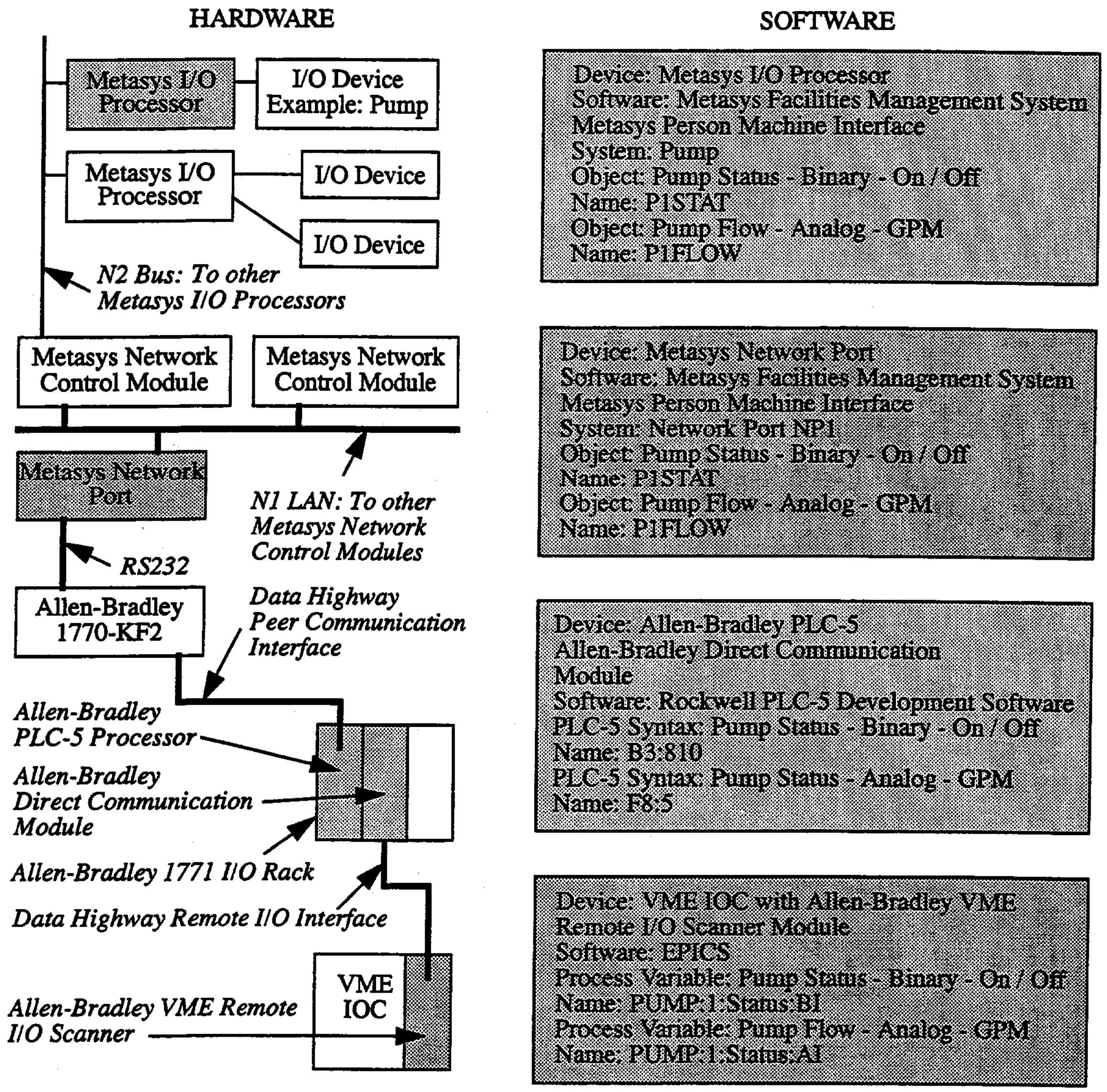
THE PLC-5 LINK TO METASYS

Since the Metasys network port emulates the communication of a PLC-5 and intercommunication between PLC-5 modules is easily accomplished with a Data Highway DF/1 protocol link, a PLC-5 module was programmed so that the data to be monitored from the Metasys network is initially gathered by an Allen-Bradley PLC-5 module. The PLC-5 was placed in a 1771 I/O chassis and connected to the network port. Since the PLC-5 processors communicate via the Allen-Bradley Data Highway and the Metasys network port communicates via an RS232 interface, there was an additional piece of hardware required, the Allen-Bradley 1770 KF2.

THE ALLEN-BRADLEY 1770-KF2

The Allen-Bradley 1770-KF2 module is a passive intelligent device which allows communication between the Allen-Bradley Data Highway and an intelligent asynchronous device programmed to handle the DF/1 communication protocol. Electrically the 1770-KF2 connects to the Allen-Bradley Data Highway in the standard multi-drop configuration, with the Data Highway station number set via DIP switches. Electrical connection to the asynchronous device is via RS232 or RS422, at data rates of up to 9600 baud [4]. The 1770-KF2 was used to electrically connect the Data Highway from the network port to the PLC-5 processor.

FIGURE 1. The METASYS-to-EPICS Link, a Hardware and Software View


HARDWARE
SOFTWARE
Metasys I/O Processor
I/O Device Example: Pump
Metasys I/O Processor
I/O Device
I/O Device
N2 Bus: To other Metasys I/O Processors
Metasys Network Control Module
Metasys Network Control Module
Metasys Network Port
N1 LAN: To other Metasys Network Control Modules
RS232
Allen-Bradley 1770-KF2
Data Highway Peer Communication Interface
Allen-Bradley PLC-5 Processor
Allen-Bradley Direct Communication Module
Allen-Bradley 1771 I/O Rack
Data Highway Remote I/O Interface
Allen-Bradley VME Remote I/O Scanner
VME IOC

Device: Metasys I/O Processor
Software: Metasys Facilities Management System
Metasys Person Machine Interface
System: Pump
Object: Pump Status - Binary - On / Off
Name: P1STAT
Object: Pump Flow - Analog - GPM
Name: P1FLOW

Device: Metasys Network Port
Software: Metasys Facilities Management System
Metasys Person Machine Interface
System: Network Port NP1
Object: Pump Status - Binary - On / Off
Name: P1STAT
Object: Pump Flow - Analog - GPM
Name: P1FLOW

Device: Allen-Bradley PLC-5
Allen-Bradley Direct Communication
Module
Software: Rockwell PLC-5 Development Software
PLC-5 Syntax: Pump Status - Binary - On / Off
Name: B3:810
PLC-5 Syntax: Pump Status - Analog - GPM
Name: F8:5

Device: VME IOC with Allen-Bradley VME
Remote I/O Scanner Module
Software: EPICS
Process Variable: Pump Status - Binary - On / Off
Name: PUMP:1:Status:BI
Process Variable: Pump Flow - Analog - GPM
Name: PUMP:1:Status:AI

THE PLC-5 AS A BRIDGE TO EPICS

In addition to the PLC-5, a 1771-DCM Direct Communication Module also resides in the 1771 I/O chassis. The DCM module provides the communication link to the APS control system with existing EPICS device support [5]. The PLC-5 is programmed with a Rockwell Software PLC-5 Development Software package. This ladder logic program reads the Metasys data which has been syntactically converted to PLC-5 binary (B3) and analog (F8) data representation by the network port via the Data Highway Peer Communication Interface. This data is then block transferred to the 1771-DCM on a time repetitive basis ensuring that the DCM module receives fresh data as read from Metasys. This repetition need only be fast enough to refresh the data every ten seconds. This is due to a time limitation on the N1 LAN data update of the Metasys network port. The DCM is then read over a Data Highway Remote I/O Protocol connection to an EPICS IOC with an Allen-Bradley VMEbus I/O scanner. The data read from the Johnson Controls conventional facilities control system may now be used in EPICS displays, alarm handling applications to warn of impending failures, and data acquisition techniques familiar to operators and accelerator physicists. Currently this interface is configured for read-only access of Metasys data. Modification for the control of

Metasys conventional facilities parameters from EPICS is easily accomplished with changes in the PLC-5 ladder logic program.

SUMMARY

Figure 1 summarizes the network which provides the link between the EPICS-based accelerator control system and the METASYS-based conventional facilities control system at the APS. It is intended that this system will provide a vital link for data acquisition and monitoring so that correlations may be made between accelerator operation and environmental parameters.

ACKNOWEDGMENTS

This work was support by the U.S. Department of Energy, Office of Basic Energy Sciences, under Contract No. W-31-109-ENG-38. Metasys is a registered trademark of Johnson Controls Inc. ARCNET is a registered trademark of Datapoint Corporation. PLC-5 is a registered trademark of Allen-Bradley Corporation.

REFERENCE

[1] Johnson Controls Inc., Metasys Interface Specification, 1994.

[2] Johnson Controls Inc., Metasys Network Technical Manual 636, Network Port Section, Technical Bulletin, 1992.

[3] W. McDowell, M. Knott, F. Lenkszus, M. Kraimer, N. Arnold, R. Daly, "Status and Design of the Advanced Photon Source Control System," *Proceedings of the 1993 Particle Accelerator Conference*, pp. 1960-1962, 1993.

[4] Allen-Bradley Inc., Allen-Bradley 1770-KF2 Reference Manual, Publication 1784-6.5.3-DU3, 1988.

[5] J. Stein, C. Seaver, G. Nawrocki, M. Kraimer, "Data Exchange from Allen-Bradley PLC-Based Systems to EPICS," these proceedings.

The Control System Database for the DØ Detector

Laura Paterno, Stuart Fuess, Stan Krzywdzinski
Fermi National Accelerator Laboratory
Batavia, IL 60510, USA

Harrison B. Prosper
Florida State University
Tallahassee, FL, 32306, USA

ABSTRACT

A central control system database has been used in the monitoring and control of the DØ experiment. The database stores all of the essential attributes of the hardware and provides on-line data acquisition processes with the access information required to read from or write to the hardware. It contains information on detector systems, low and high voltage power supplies, cryogenics monitoring, argon purity monitoring and environmental conditioning. The database serves as the master copy of local databases which reside in front-end computers. It is a relational database, implemented using DEC VAX Rdb/VMS. Utilities were provided for updating and accessing the database, either interactively or in a batch mode. To speed up access in the on-line environment, a read-only copy of the database is maintained and updated every day; detached server processes with sufficient resources provide processes with fast uninterrupted read-only access. Merits and drawbacks of the present system are discussed along with possible enhancements for the future.

1. INTRODUCTION

The DØ detector [1] at the Fermi National Accelerator Laboratory was constructed to study proton-antiproton collisions at a center-of-mass energy of 2 TeV. The detector consists of three nested shells. The innermost shell contains the central tracking and transition radiation detectors; surrounding these is the calorimeter, and surrounding the calorimeter is the muon detector. Combined these detector subsystems consist of ~100,000 channels. The monitoring and control of the channels of the detector as well as collection of physics data was accomplished through the DØ data acquisition system (DAQ), which was designed with two independent data paths for communication [2]. We will not discuss the physics data path in this paper.

Because of the large number of channels to control and monitor, a mechanism was needed to keep track of all the information for each channel: the DØ control system database. The database, known as the Hardware database (Hdb) [3,4], is a relational database implemented using DEC VAX Rdb/VMS. It was designed to provide access path descriptions to Control Data Acquisition (CDAQ) [5] processes as well as nominal settings with their associated tolerances, and descriptions of each device. Currently, the database contains ~5,000 devices which describe the channels of the detector.

The information from the database is downloaded to front-end processors which are either Motorola 68020 processors in VME crates or IBM PC processors. Each processor has its own local database which is updated from the master Hdb database. The local databases contain the access information and alarm conditions for all the devices that the processors monitor and control. The processors use the alarm information to monitor the hardware for failures and notify the on-line data acquisition system when changes of state occur in the system, that is, when a device enters into or leaves the alarm conditions.

In the following sections we discuss the organization of the Hdb database, the utilities used to maintain and access it, the merits and drawbacks of the current database and future enhancements under consideration for the 1999 run of the DØ detector.

2. DATABASE ORGANIZATION

2.1 *Structure of the database*

The database is relational in form and has a variety of capabilities that the DEC Rdb/VMS environment provides (concurrency control, data integrity, security, data independence, a data manipulation language, query optimization, remote access, etc.) as well as management utilities (RMU, RdbExpert, DECTrace). The database maintains data in a two-dimensional tabular format with 'flat' computer files. A table (or relation) contains a number of rows (or records) which are instances of table elements; the rows consist of columns (or fields) which are data elements and represent attributes of the table.

The Hdb database, as seen in the Entity-Relationship diagram in Figure 1, currently contains 39 tables which describe the hardware devices and their relationships in the detector. The boxed area contains the tables which hold all of the access and alarm condition information for the hardware devices. The other tables are used to describe relationships between the devices.

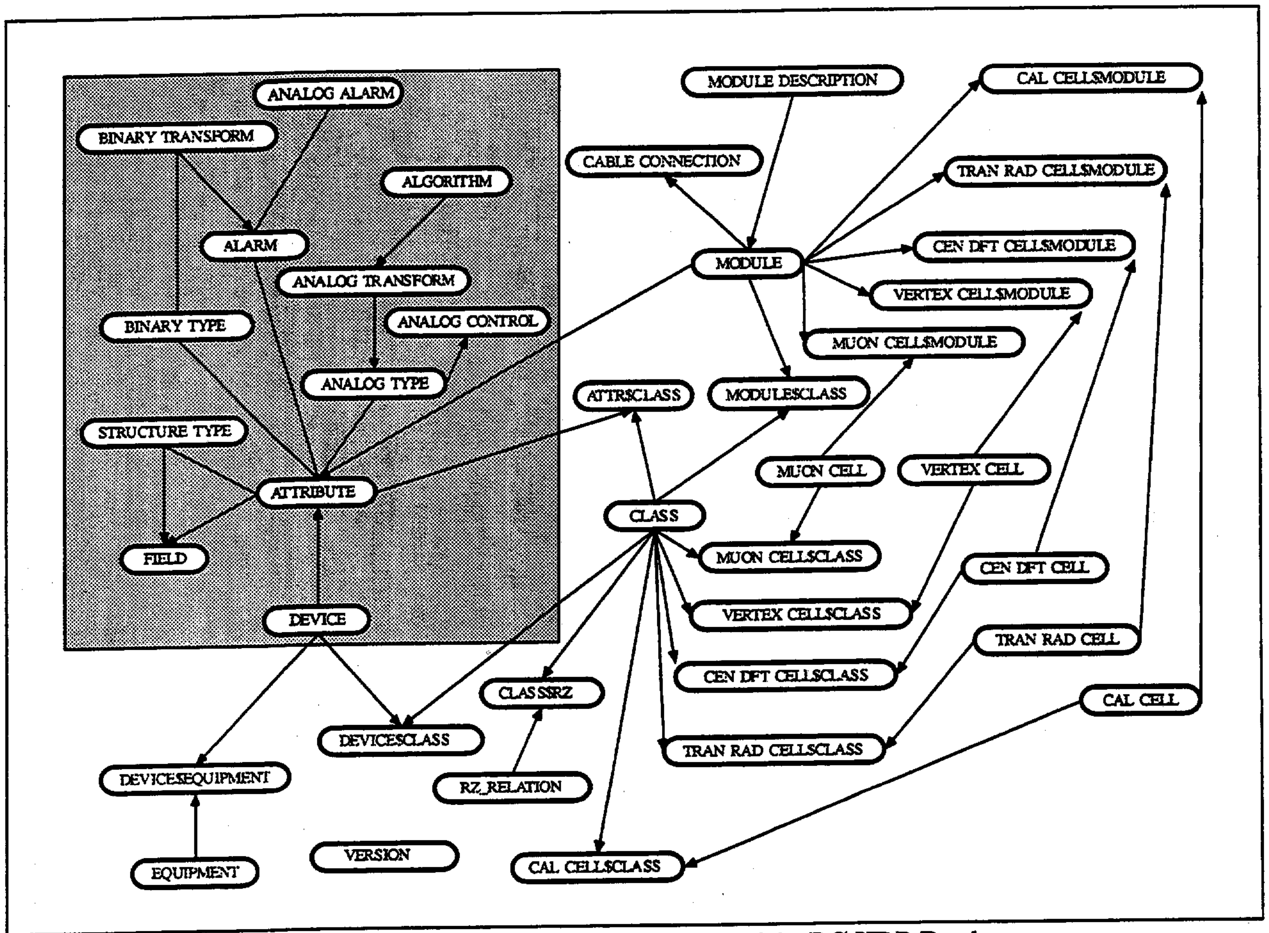


Figure 1 - The Entity-Relationship Diagram of the DØ HDB Database

The individual boxes represent the tables of the database and the connecting lines represent the relationships between the tables. A line with no arrows indicates a one-to-one relationship; a line with arrows means a one-to-many relationship. For example, a device can have one or more attributes associated with it but an attribute can have only one alarm associated with it.

2.2 *Devices*

Central to the Hdb database is the notion of a *device*. A device may represent any physical hardware object in the detector or in the on-line software. The device may contain any number of monitoring and/or control channels. For example, a power supply is a device because it has voltage or current output which can be monitored and controlled. Only the access information necessary to read and/or write to the physical device is represented in the database. It can (and does) contain some default values used to control the device.

Distinct properties of a device are represented as *attributes*. An attribute may describe analog or binary monitoring and control points of the device. It contains access information which enables CDAQ processes to request readings or settings to devices through the front-end processors. They also have associated alarm information which is loaded into the front-end processors. Devices and their associated attributes were carefully named to provide an easy association between the database objects and the physical devices they represented.

Many of the DØ detector electronics systems contain the same kinds of physical devices (power supplies, temperature monitors, etc.). The monitoring and control channels for these devices are described in the database using structured attributes [6]. The common data for the similar devices was placed into a set of 'elementary devices', which were only entered once. These elementary devices are then referenced by structured attributes of a device through the field table. This made the database more modular in structure and required less data to be entered into the database.

3. UPDATING AND ACCESSING TOOLS

3.1 Updating Tools

Three programs were used to update the Hdb database: HDBatch [7], HDBrowser [8], and HDB Entry [9]. They were written in C or FORTRAN with either embedded RDML or SQL statements used to access the database. The programs provided a variety of services to their target clients including browsing, listing, entry, modification, and deletion of the devices in the database.

The initial loading of the database was done using HDBatch. This utility uses text files which may be created with the HDBedit command which invokes the VAX Language Sensitive Editor (LSE) with a special environment which contains all the HDBatch commands. The files described the devices to be loaded into the database and HDBatch updates the database either interactively or through a batch job running under the VAX/VMS operating system. The utility was written in C and calls low level access routines [4,10,11] to create, delete, modify and list requested devices.

The HDBrowser program uses the VAX Forms Management System (FMS) to display forms whose fields represent data fields in the database. Because the program is interactive, users can see what they are currently entering or have previously entered into the database in a layout which is easy to understand. The program also uses the same low level access routines as HDBatch.

The HDB Entry program was designed to allow easy creation of multiple devices of identical type. It was written in FORTRAN with embedded SQL statements. A portion of the code was generated from the structure of the database. In other words, if the database was changed, the code which dealt with the actual access to the database could be regenerated automatically. The program has both an interactive and batch mode interface. The interactive mode was written using the VAX Screen Management facility (SMG).

3.2 Database Access Utilities

There are two layers of routines provided to access the Hdb database. The first consists of low level routines: create, delete, modify and view procedures for each table in the Hdb database. The routines exist in a shared image library. Thus, if changes need to be made to the implementation of any routine, all clients of that routine automatically get the changes without recompiling or relinking their code. All of the routines are used by the HDBatch and HDBrowser programs; only the view routines are used by the second layer of access routines.

The second layer of routines is called the Database Services (DBS) package [12]. The DBS routines were written in PASCAL with calls to the low level C routines. They were designed to make it easy for the CDAQ processes to get the information they need. There are also a few routines used by non-CDAQ processes to obtain descriptions of the devices in the database for the purpose of displaying that information. This has helped experimenters to diagnose hardware failures that have occurred during data taking. The Hdb server processes described below also use these routines.

4. SERVER PROCESSES

During the early running of the DØ detector it was discovered that access times to the database were slow. Client processes which downloaded detector constants to the front-end processors often doubled their normal execution times. The problem was traced to writers making frequent modifications to the database. They effectively locked out all processes interested in just obtaining information from the database while they were writing. Fast access to the database also necessitated processes having more system resources and quotas available to them.

Two improvements to the system alleviated these problems and reduced program execution times to acceptable levels. Since restricting modifications to the database was not an option, a read-only copy of the database was created and then updated daily. It solved the problem of writers affecting access times and also served as a natural backup to the master read-write database. The second improvement was the creation of the Hdb server processes. The servers are detached processes which have the necessary resources and quotas. They keep the database permanently opened to clients. Normal DBS access routine calls in a client application are replaced by ones which send messages (via an InterTask Communication (ITC) package [13] layered over DECNET) to the server. The server performs the database access and returns the results to the client process.

5. APPLICATION PROGRAMS

5.1 Downloading

Before data taking (running) commences, the detector control system must be downloaded with the appropriate hardware device settings. A master synchronizing program accesses configuration files that describe the run parameters and converts them into a sequence of commands, interpreted by multiple instances of the downloading program which execute in parallel. The downloading program parses the commands and accesses the database through CDAQ routines to perform the settings. Typical cold-start downloading times average 7 to 9 minutes. Run-to-run changes can be performed in less than 3 minutes.

5.2 Monitoring and control

There are a variety of programs which perform general utility or subsystem specific monitoring and control functions. Many of these programs access the database through CDAQ services. Examples include: (1) a program to periodically monitor a set of devices in the detector and log the readings for later trend studies, (2) programs to monitor and control various power supplies, (3) low level programs which monitor and access any device and display the readings as a bar graph or a time trend on a strip-chart graph, and (4) a program that synchronizes the triggering of the detector. Another program, which displays significant event changes in the system, accesses the database directly through the DBS package.

6. ADVANTAGES AND DISADVANTAGES OF THE CURRENT DATABASE

The current database system provided very stable, reliable, and efficient service through the 4 years of DØ running. No data corruption occurred during this entire period which can be attributed to the use of a well documented and supported commercial database. The database server process enabled clients to get quick, read-only access to device information without the need for additional resources in order to obtain fast access times. This helped to improve the downloading times for the DØ detector and reduced the amount of physics data which would otherwise have been lost if downloading times were slower. The use of both a read-write and read-only database enabled writers to update the database without slowing down the readers of the system. This also improved the downloading time as writers may block out access to readers while updates are going on.

There are of course disadvantages to the system as well. Only 12 out of the 39 existing database tables are actually used. This was because not enough manpower was available to update them and the need for the other information was never critical. The database updating programs also have flaws. They are either too slow (HDBatch and HDBrowser), the interface can only be understood by an expert (all), or a full set of services was not provided (HDB Entry).

7. FUTURE CONSIDERATIONS

The present system has been reliable due to the capabilities and data integrity provided by a commercial database product. Experience with non-commercial databases was not always positive. The products experienced data corruption problems and required much more manpower than the commercial solution. A commercial database package with all the present features of DEC Rdb will be a critical requirement for the future database.

The DØ collaboration is considering what upgrades can be made to the experiment. The upgraded detector will have approximately 10 times more channels to monitor and control. This puts strong requirements on database access times to ensure fast downloading of all detector constants. Server access to the database for time-critical clients will be essential in view of the success with the present database server and our decision to switch to a multi-platform environment.

The database itself will be restructured in line with to the new detector subsystems. These new systems will have thousands of identical channels with only slightly different address access. This will require rewriting all the database updating and accessing software. A single integrated tool that is intuitive to the users (hardware experts) and not modeled on the database definitions would be preferable. The tool should be written in a single language with one database query language embedded for ease of maintenance. It will also have to run on the different platforms that DØ supports.

ACKNOWLEDGEMENTS

The authors wish to acknowledge the considerable contributions of the members of the Accelerator and Computing Divisions of the Fermi National Accelerator Laboratory as well as those of the DØ Collaboration.

REFERENCES

[1] S. Abachi, et al., (Fermilab PUB93-179-E) Nucl. Instr. and Meth. **A338** (1994) 185.
[2] S. Ahn, et al., (Fermilab PUB93-324-E) Nucl. Instr. and Meth. **A352** (1994) 250-253.
[3] M. Frey and A. Waller, DØ Hardware Database, DØ Note 821 (1988) unpublished.
[4] M. Frey and A. Waller, DØ Design Notes-The Hardware Monitor Database, DØ Note 824 (1988) unpublished.
[5] J.F. Bartlett, S. Fuess, & H.B. Prosper, User's Reference Manual for the DØ Control Data Acquisition Services, DØ Note 866 (1988) unpublished.
[6] S. Krzywdzinski, Structure Type Devices in HDB Database, DØ Note 1279 (1991) unpublished.
[7] A. Waller, HDBatch: Batch Entry Program for the Hdb Database, DØ Note 915 (1990) unpublished.
[8] M. Frey, HDBrowser: Interactive Entry Program for the Hdb Database, DØ Note 879 (1989) unpublished.
[9] L. Paterno, H.B. Prosper, & R. Raja, The DØ Hdb Entry Program, DØ Note 1512 (1993) unpublished.
[10] M. Frey, Hdb Database User's Guide - C version, DØ Note 822 (1988) unpublished.
[11] M. Frey, Hdb Database User's Guide - FORTRAN version, DØ Note 823 (1989) unpublished.
[12] S. Krzywdzinski, User Reference Manual for HDB Database Services, DØ Note 1280 (1993) unpublished.
[13] J. Featherly, InterTask Communications package (1989) unpublished.

A Generalized Correlation Plot Package for the CEBAF Control System*

D. Wu, W. Akers, S. Schaffner, H. Shoaee, W. A. Watson, D. Wetherholt
Continuous Electron Beam Accelerator Facility, Newport News, VA 23606 USA

ABSTRACT

The Correlation Package is a general facility for data acquisition and analysis serving as an online environment for performing a wide variety of machine physics experiments and engineering diagnostics. Typical correlation experiments consist of an initial set of actions followed by stepping one or two accelerator parameters while measuring up to several hundred control system parameters. The package utilizes the CDEV [1] device API to access accelerator systems. A variety of analysis and graphics tools are included through integration with the Matlab math modeling package. A post- acquisition script capability is available to automate the data reduction process. A callable interface allows this facility to serve as the data acquisition and analysis engine for high level applications. A planned interface to archived accelerator data will allow the same analysis and graphics tools to be used for viewing and correlating history data. The object oriented design and C++ implementation details as well as the current status of the Correlation Package will be presented.

INTRODUCTION

The tasks of commissioning and optimizing the performance of an accelerator require performing many machine experiments involving the acquisition and online analysis of a large amount of data. This data is usually obtained from a diverse and disparate array of devices, instruments and processes. Each source of information or data may have its own unique acquisition characteristics or control idiosyncracies, e.g., the required settle time between changing a magnet setpoint and reading a downstream BPM, or the synchronization of the motion of a wirescanner with successive beam samplings. Additionally, most experiments involve a myriad of activities which extend beyond simple data acquisition. This includes complex instrument and experimental setup prior to data taking, monitoring of some control system parameters for conditional data acquisition (e.g. recording BPM data only if sufficient beam current is present), intermediate evaluation of the results and finally a post experiment set of activities including analysis, graphics, and data archiving.

The objective of the development of the Correlations Facility has been to provide accelerator users with an integrated environment for performing a large array of machine physics and engineering diagnostics experiments with uniform access to practically all available machine data. The generalized data acquisition capability of this package provides automatic and transparent access to new data types and sources as they become available within the control system, without the need to rebuild any components of the correlation software. Similarly, within this environment one has the flexibility to automatically vary any system parameter that can be manually adjusted via the control system.

Additionally, by designing the system around a data acquisition engine, we have provided the capability for using this engine in any high level application or automation procedure, such as emittance measurement or cavity gradient calibration, which requires controlled data acquisition and analysis.

The Correlation Package was inspired by SLAC's Correlation Plot software[2] and provides many of its features. However, its implementation and capabilities reflect the advances in the fields of software engineering, graphics, user interface and math modeling since that pioneering package was developed.

PRINCIPLES OF OPERATION

A typical accelerator physics experiment consists of several steps of the following nature:

- A *pre-experiment setup* stage which may include turning on devices or facilities, calibrating, status checking, etc.
- A *control* stage involving varying the setpoint of one or many devices over a desired range.
- An *interim* period between control and data acquisition when the experimenter may perform a set of tasks or actions including waiting for a length of time (e.g., for the transients in the control device to die out), checking the status of any control system signal, executing some stand-alone programs or scripts (e.g., to initialize or prepare the data acquisition equipment), and finally, deciding to proceed or to abort based on certain observed conditions.
- A *data acquisition* step whereby up to several hundred control system signals are sampled and recorded.
- A *post experiment set of activities* including analysis, graphics, resetting devices, saving results in a database, etc.

To accommodate these requirements for an integrated package, the correlation facility has been implemented in terms of several key concepts or module:

- A **control module** capable of stepping the setpoints of one or many control system devices, such as corrector strengths, cavity phases, or feedback setpoints.
- A **data acquisition module** for recording any available control system signal such as beam orbits from BPMs, RF parameters, feedback measurement and control parameters.
- An **action module** capable of performing a variety of tasks including synchronized and time delayed data acquisition, executing stand-alone programs or scripts, evaluating conditions based on machine and beam states for continuing or aborting an experiment, and performing many control system actions such as calibrating BPMs.
- A **graphics module** for displaying any sampled data against any sampled or stepped variable. In the near future, this module will also perform a variety of signal analysis and curve fitting functions on the sampled data.

To facilitate using the correlation package as an embedded engine in high level applications requiring controlled data acquisition, we developed a portable **Correlation Engine** encapsulating the *control, data acquisition* and *action* modules. Figure 1 illustrates the functional components of an experiment as performed with the Correlation Engine.

Steps of a correlation experiment:

- Perform a set of pre experiment actions which may include turning on devices or facilities, calibrating, status checking.
- Step the setpoint of a list of devices over a desired range, each optionally preceded and/or followed by a set of actions including specified time delays (settling time).
- Perform a set of actions between stepping and sampling, including time delays, status checking, executing programs or scripts (e.g., to initialize the data acquisition equipment), and finally, deciding to proceed or to abort based on certain observed conditions.
- Record a list of control system signals, each optionally preceded and/or followed by an action list.
- Perform a post experiment set of activities including analysis, graphics, resetting devices, saving results in a database, etc.

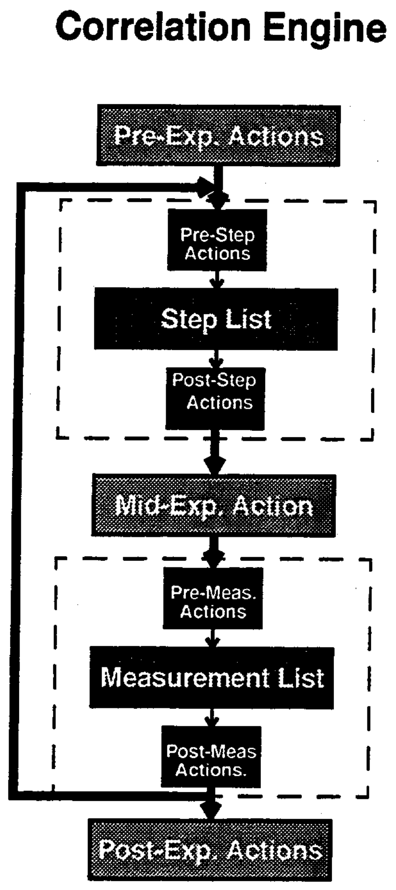


Figure 1: Anatomy of A Correlation Experiment

DATA ACQUISITION INTERFACE

XACT (X-based Accelerator Correlation Tool) is an X11/Motif based graphical user interface (Figure 2) for performing correlation experiments. It is the first application software using the Correlation Engine as an embedded facility. From this interface one may specify *step* and *sample* variables, *action* lists and other acquisition and control parameters. It is also a control panel for starting and stopping experiments as well as saving the experimental setup and the acquired data to files for later recall and analysis.

Device Selection

All signals names specified as step or sampled variables are validated against the control system. Validation can be toggled on and off through the pull-down "Option" menu. One may also use wild cards in specifying signal names in which case a list of all matching signals will be presented. You may choose one, many, or all available signals.

Step and Sample Variables

Users may select up to two step variables although the Correlation Engine allows unlimited number of step variables. Available variables include all control system signals including

- magnet setpoints
- RF setpoints
- Feedback parameters
- Multidevice knobs (composite devices)

A special device keyword, INDEX allows sampling signals without stepping any device. Selecting INDEX as step variable coupled with time delays between samples (entered via action commands), provides a flexible facility for recording the time history of signals. Currently, a stepped device follows a ramp function with the user specifying the number of steps, and the initial and final values of the ramp. Sinusoidal, square and triangular input functions will be introduced in future.

Entering sample variables is similar to the step variables and up to 1000 may be selected. A special sample variable, INPUT, allows manual data entry for each step.

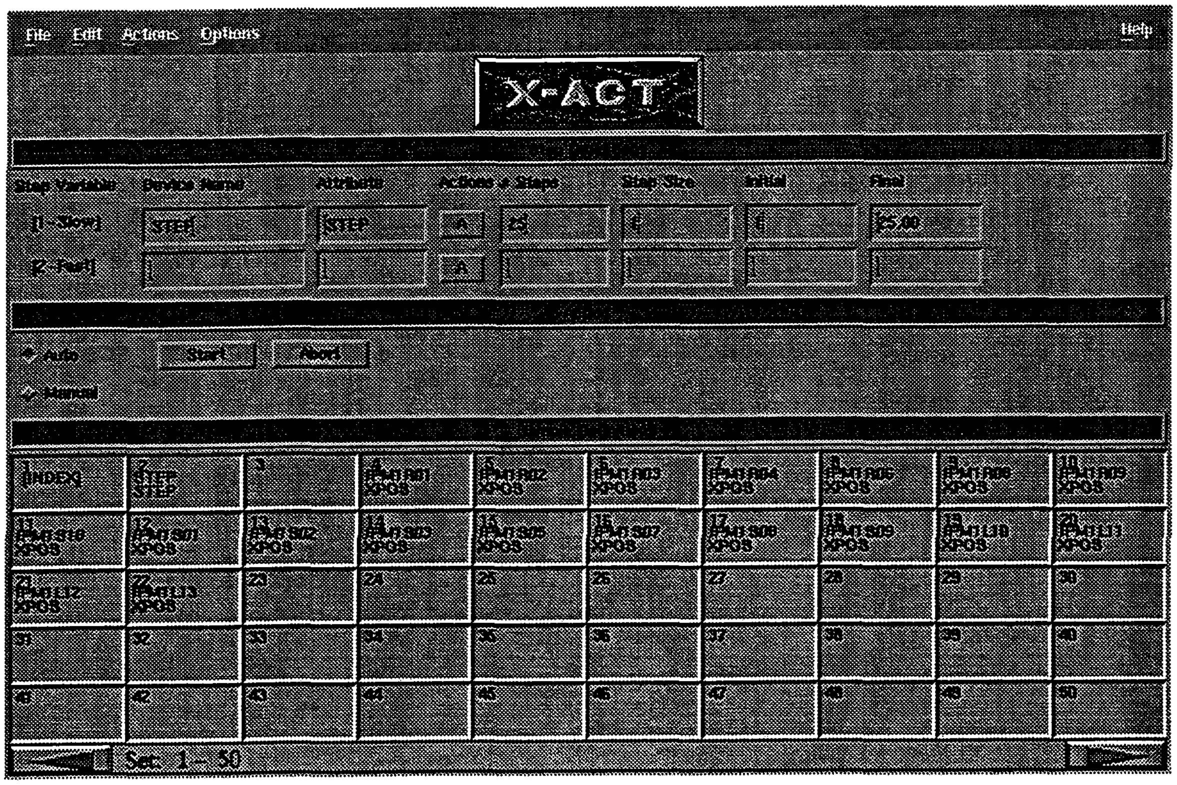


Figure 2: Data Acquisition Interface for the Correlation Package

Actions

For each step and sample variable one may specify a list of pre and/or post actions. Additionally, Initial (pre experiment), Intermediate and Final (post experiment) actions are also available from the Actions menu. Currently available actions include "Time Delay" (simple time delay after stepping or sampling), "Time Delay Sync" (synchronized sampling) and "Script" (executing a shell or a Tcl script at any point during the experiment).

Control

The data acquisition may be performed in auto or manual form. In manual acquisition the program acquires a sample only after the user has so indicated. Users may also abort the acquisition in which case they will be prompted whether the data should kept or discarded.

Once an experiment has been defined, the entire setup including the list of sample variables, step variables and actions may be saved for future recall and execution. This allows for setting up and debugging complex experiments in advance of actual use in the control room. Additionally, after an experiment, one may save the collected data for analysis and plotting with the Matlab analysis package.

ANALYSIS

The analysis portion of the Correlation Package is written using the Matlab [3] modeling package and is called the Matlab Analysis and Graphics Laboratory Environment (MAnGLE). Matlab was chosen as the basis of the analysis portion because it contains a wealth of graphing and analysis functions and because most laboratories have Matlab licenses. MAnGLE presents the user with a graphical interface to the data collected by XACT allowing them to perform basic analysis and graphics functions by pointing and clicking. Communication between MAnGLE and XACT is through disk files. MAnGLE reads the XACT setup file to obtain a list of stepped and sampled signals as well as information about the input functions. XACT stores the data collected in a file specifically formatted for Matlab. In the current implementation, the user must save the XACT setup and data file and then start up MAnGLE as a separate process and select the named data files for analysis. In the future, XACT will spawn MAnGLE and users will be able to analyze and view the data as it is being collected.

The MAnGLE user interface is divided into two sections. The first is a control panel which displays the device/attribute pairs the user selected with XACT along with menus and associated popups which allows the user to select predefined analysis functions and options. Graphics and analysis results are displayed in a separate window. The current implementation of MAnGLE provides 2-D graphics display of any correlation variable against any other. There are currently three plot options. The entire data set may be displayed at once in a scrollable window. A second plot option allows selecting a default X variable and then plotting any other variable as the Y variable. Y variables are selected by clicking on the device/attribute name displayed in the control panel window. The graphics window can be cleared between plots or plots may remain in the window to display a family of curves. A third option is to select a default Y variable and cycle through any number of X variables. Future plans include bar chart displays and 3-D graphics displays. Analysis functions such as curve-fitting, histograms, minima and maxima determination and FFTs will be added later. Future implementations will also allow users to write their own Matlab scripts (m-files) which can be added to MAnGLE.

IMPLEMENTATION

The correlation Plot Package has been developed utilizing Object oriented design and C++ implementation. The core of the package which may be embedded as a data acquisition module in any high level application, consists of the correlation specification, *corrSpec,* and the correlation *engine. CorrSpec* maintains all the information that defines an experiment; the correlation engine makes use of this information to run through the experiment.

Correlation specification consists of five major parts: a set of initial actions, a set of step variables, a set of intermediate actions, a set of measurement variables and set of final actions.

From an object point of view, a *corrSpec* object may hold lists of step objects, action objects and measurement objects (Figure 3). Each step or measurement object could optionally contain lists of pre-actions and/or post-actions as needed. The *corrSpec* object also contains all the necessary methods to manipulate these lists of objects.

A step object can vary any accelerator parameter with a given initial value, step size, final value, and a stepping function (ramp, square, triangle, sine). Step specifications may also be time or an index. In the case of an index, the engine collects all sample variables sequentially without stepping any device in between the samples. An *action* object can be any general action, such as a time delay (synchronous or asynchronous), adjusting any control system parameter, conditional action by comparing some parameter values, waiting for some events to take place, or executing a script. A measurement object is just a container which holds the measured data.

All classes including step, action and measurement, are organized in a hierarchical structure so that a simple interface can be presented. A major benefit of this organization is the ability to augment the action class without the need to modify the access interface.

The correlation engine's interface to the accelerator environment is solely through the CDEV device layer and is thus independent of the underlying control system. This allows any correlation application to have uniform access to any machine data regardless of its source, such as online data from low level microprocessors, static data from a central database, modeling information, etc. Similarly, once a new device (or device type) has been added to a service under CDEV, such as EPICS [4], it becomes automatically available to correlation applications for measurement and/or control.

The correlation engine is also capable of periodically calling user callback functions to allow applications to perform other tasks such as temporarily suspending data acquisition or aborting the experiment.

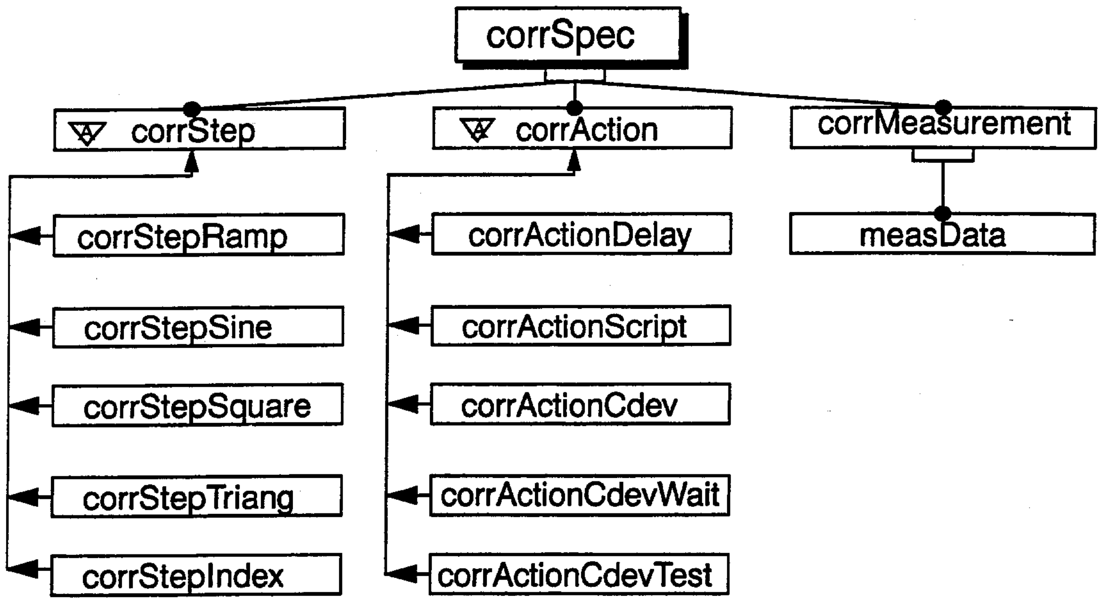


Figure 3: Correlation Plot Package Object Classes Scheme

FUTURE PLANS

The correlation package which is currently undergoing beta testing contains the basic features that are necessary for online data acquisition and control. In order to provide a comprehensive environment for performing online accelerator physics studies, the following developments are planned:

- Uniform access to all control system data including correlation, archived history, and high speed buffered data;
- Complete online data analysis capability including curve fitting, signal processing, and statistical analysis;
- Availability of graphics and analysis directly from XACT, and transparent data transfer from XACT to MAnGLE;
- Availability of a variety of input functions such as triangular, square, and sinusoidal;
- Implementation of correlation engine as a service under CDEV. This would allow its use as an embedded engine by high level applications such as emittance calculation and lattice diagnostics.
- Implementation of a scripting language to automate all phases of an experiment including data analysis.

REFERENCES

[1] J. Chen, G. Heyes, W. Akers, D. Wu, W. A. Watson III, *CDEV: An Object-Oriented Class Library for Developing Device Control applications,* International Conference on Accelerators and Large Experimental Physics Control Systems, October 1995.

[2] L. Hendrickson, N. Phinney, L. Sanchez-Chopitea, *Correlation Plot Facility in the SLC Control System*, Proceedings of Particle Accelerator Conference, May, 1991.

[3] Matlab reference Manual, Copyright The MathWorks, Inc., 1984-1993

[4] Leo R. Dalesio, et. al. *The Experimental Physics and Industrial Control System Architecture: Past, Present, and Future,* International Conference on Accelerators and Large Experimental Physics Control Systems, October 1993.

Database Requirements for the Vacuum Systems of LHC

M. Steffensen and P. M. Strubin
CERN, Geneva, Switzerland

Abstract

The new Large Hadron Collider (LHC) which will be built over the next decade at CERN requires two independent vacuum systems. One will be a high vacuum for the two 26 km long beam pipes and the other is needed for the insulation vacuum in the cryostat and helium distribution system for the superconducting magnets. Several thousands of components will be acquired, assembled and installed in the ring tunnel. All active components will eventually be remotely controlled. Several phases can be identified during the project lifetime, starting from the preliminary design and ending with exploitation. Each phase requires different sets of data to be available. This paper identifies the required data, when this data should be available, the needs for integration with other equipment or project databases and the tools which are felt necessary to manage the data.

1 INTRODUCTION

A considerable amount of information has to be provided, stored and exchanged during the various phases of a large project, like the LHC accelerator. Powerful database management systems are now available and ease the organisation and storing of this data. But retrieval or manipulation tools are as important as the database engine itself.

The required information varies during the project lifetime, as does the necessity to share this information. This paper tries to identify the information required to design, produce, install and eventually commission the vacuum system of LHC. It will also focus on those areas where information may not be available in the most suitable way.

2 CONCEPTUAL DESIGN PHASE

The functionality of the equipment must be defined properly during the conceptual design phase, but the components of the various systems, including the vacuum system, are only loosely defined. The aim at this stage is to provide a feasibility study, along with cost and resource estimates. This activity is carried out based on experience with previous accelerators, prototypes, manufacturers' catalogues, etc. In the case of the vacuum system, layout sketches are the primary input for the topology. From these, the vacuum specialists evaluate requirements for pumps, valves and measuring equipment. Adding them all together will provide a cost estimation.

During this phase, it is important to have access to the design and manufacturing data of previous accelerators (e.g. LEP), as well as to up-to-date supplier documentation. At CERN, design and manufacturing data from previous accelerators exists mainly in the form of paper reports and drawings, the latter often being stored in EUCLID or AUTOCAD format. Some manufacturing data for vacuum chambers exists in the form of ORACLE tables. On the other hand, manufacturers' catalogues are beginning to be published on electronic media, such as CD-ROM disks, which can be accessed from desktop computers.

At this stage, sharing of data is mostly done during formal and informal meetings. Minutes of these meetings can (and start to be) published by electronic means, like the World Wide Web. There is little need for a central database and in most cases a good spreadsheet program is an adequate tool to rapidly evaluate the cost of various solutions.

However, at the end of the conceptual design phase the cost and resource estimates should be stored in a central database, as they are key components for the planning and the follow-up of the budget and expenditure.

3 DETAILED DESIGN PHASE

The requirement for storing and sharing data considerably increases during the detailed design phase. The primary input data is the layout of the accelerator, which is derived from the conceptual design and gets more and more refined. The level of detail required is however varying strongly, depending upon the usage. For instance, the vacuum specialist responsible for distributing the pumping equipment needs to know about possible interferences with other equipment, whereas the controls team only needs to know the location of the equipment for cabling.

The layout data must allow for version handling, as it must be possible to work on several different implementations simultaneously. A good example of the implementation of such a database is given in [1]. It has been widely used by the Vacuum Group for the LEP accelerator. However, there is a strong need for better tools to access and manipulate the data.

Layouts are best represented on drawings rather than lists of equipment. The main reason for this is that these drawings are presented to various decision committees, where discussions in front of a computer screen are not easily possible. Another, almost trivial, reason is that the human eye often points to possible interferences much quicker than sophisticated analysis programs. Modifications of layouts should be possible using tools which resemble object drawing programs rather than the present method of ORACLE Forms used by Vacuum Group.

The objects themselves are drawn by the design offices using CAD tools (mainly EUCLID and AUTOCAD at CERN) and should be stored with various levels of details in a central database. The relations between the objects also have to be stored in a database. Work is going on in this field in the Vacuum Group and elsewhere at CERN [2], aiming at a generalised approach for defining and grouping objects.

It should be possible to access and manipulate these objects on a screen using essentially a "point and click" approach. For instance, the vacuum specialist in charge of placing the pumping elements would display a section of the layout of LHC and add or displace the pumps, the model of which would have been drawn by the design offices or extracted from manufacturer's documentation, when available. Again by analogy with common drawing programs, the layout should be drawn in layers corresponding to the various equipment groups.

Not all data has to be entered manually into a database. A large accelerator like LHC has a highly repetitive lattice for more than three quarters of its circumference. Hence automatic programs can be used to enter data like pump locations and the associated cabling. This approach was successfully used for LEP, where the initial cabling and routing had been produced automatically from the available description of the vacuum system in an ORACLE database. Similarly, pumps, gauges and roughing valves have been automatically placed using a FORTRAN program, the input of which was again the description of the vacuum system extracted from the database.

A large amount of experimental data will be produced during the detailed design phase. Several experiments are run to evaluate the effect of synchrotron light and the way to cope with it in a superconducting environment [3]. This data should be stored, but do not necessarily have to be in central database, as it is highly specialised.

As the volume of data increases considerably during the detailed design phase, it becomes necessary to introduce global data management tools, often referred to as Engineering Data Management Systems (EDMS). Commercial products start to be available and used in industry. An EDMS Task Force has been set up at CERN to try to use such a product for the design of LHC experiments [4].

4 MANUFACTURING OF COMPONENTS OF THE VACUUM SYSTEM

The Vacuum Group's involvement during the manufacturing process is mainly related to the tendering process, the follow up of the fabrication and the acceptance tests. At this stage all possible critical paths should be identified. Planning data must therefore be easily available and updatable. A solution to this problem is proposed in [5].

During the tendering process, detailed drawings must be made available from the design offices. In general, these drawings are produced using CAD tools but they still are output on paper, although more and more industries use CAD programs with an interchangeable output format, like AUTOCAD. An up-to-date supplier database is also a key ingredient for successful tendering, in particular to minimise the need for time consuming Market Surveys.

During the follow up of the manufacturing, critical data, such as tolerances, must be obtained from the manufacturers and stored in a database at CERN. During the construction of LEP, manufacturing data of the lead cladding process were downloaded from the supplier in Germany over telephone lines and stored in ORACLE tables. It is also at this stage that inventory data, like serial numbers, start to appear. Finally, the results of the acceptance tests must also be stored. The approach proposed in [2] seems adequate to relate the data properly with the different objects being manufactured. However, here again, a considerable effort to ease data input and retrieval has still to be done. In particular, it should be avoided as much as possible that data produced at the manufacturer's place has to be printed and later typed into a database.

5 INSTALLATION AND COMMISSIONING

During the installation phase, the primary data required is layout, planning and stock keeping. During this phase, there is little input from the Vacuum Group, except for delivery time of components and feedback from installation teams. But there is a considerable need for planning and logistics information.

The location of every component, identified by some serial number, should be recorded during installation. This data can be of great use when defects appear which could be linked to a particular series or production time of some components, requiring systematic repairs or modification. An attempt to store the location of vacuum equipment was started in LEP, where bar code labels were glued on the vacuum chambers prior to installation. A programmable bar code reader was used to scan parts of the accelerator after installation and this data was then loaded over RS-232 lines into the database computer. Unfortunately, the scan was never finished and the data has not been used. There was no technical problem, however.

The quantity of data to store increases again when the time comes for commissioning. At this stage, the main control system for LHC should be available. Hence, this data could be automatically fed into the database using data logging and archiving programs. This data is crucial for the future maintenance activities of the accelerator. Because of the tight time schedule of the installation work, some minor defects may have to be left for a later repair. The commissioning data would allow for a map to be produced of the temporary repairs which have to be fixed at the first possible occasion.

6 CONTROLS AND OPERATION

The specific database requirements for controls are multiple and start during the detailed design phase. Initially, the cabling of all remotely controllable equipment must be defined, relying on the layout data and properties of the equipment. As already mentioned, a large part of this activity can be done using automatic programs. In the exploitation phase, however, when only incremental modifications are done to the accelerator, a graphical approach is preferred.

An important effort has been made to provide operational models for the vacuum equipment [6]. These models are best described using object oriented notations. Thus, an object oriented database is likely to be the most efficient repository for the description of the properties and behaviour of the vacuum equipment. Work is presently going on in the Vacuum Group to assess to what extent a conventional relational approach could be used until commercial object oriented databases really exist. The messages and the required parameters used to drive the equipment must also be stored in the database. This has been done for LEP and allows for a coherent behaviour of the equipment, as well as for automatic decoding of the messages [7].

Finally, as soon as part of the accelerator come into operation, the pressures should be logged. Unlike all previous databases, this requires an on-line database, with update or insert times compatible with the physical data acquisition. Extracting data can be done off-line, however. Data reduction and archiving algorithms should be made available as soon as some data is available from the vacuum system via the main control system. A typical logging system should gather data at a relatively high rate (e.g. one reading every few minute). This data must remain accessible for a period of time of the order of one or two weeks, after which only a subset should be stored and kept for the lifetime of the accelerator.

7 CONCLUSIONS

The presently used Data Base Management System, ORACLE, is adequate for the storage of the huge amount of information required for the design, construction and commissioning of the LHC accelerator. Most effort will have to be put in the data manipulation tools, in particular to allow for a more graphical approach. A significant effort will also have to be put into the Configuration Management and into the global management of all this data, using the emerging commercial Engineering Data Management Systems.

REFERENCES

[1] CERN's Accelerator Description Using Databases, A. Albrecht et.al., proceedings of the 9th International European ORACLE User Group Conference Cannes, 1992, CERN SL/92-19 (OP)
[2] J. Schinzel, private communication, CERN-1994
[3] R.Calder at al., Synchrotron Radiation Induced Gas Desorption from a Prototype LHC Beam Screen at Cryogenic Temperature, CERN AT/95-42 (VA), LHC Project Note 7
[4] C. Hauviller, chairman of EDMS Task Force, private communication, CERN-1995
[5] A Planning and Scheduling System for the LHC project, G. Bachy et al., LHC Note 355
[6] Operational Protocols for Vacuum Systems, G. Baribaud at al., CERN AT-VA (91-06)
[7] Automatic decoding for the control messages for the LEP Vacuum System, P.M. Strubin, proceedings of the 8th International European ORACLE User Group Conference, Madrid, 1990, CERN AT-VA (90-93)

Personnel Safety System Status Reporting via EPICS at the Advanced Photon Source*

S. J. Stein
Experimental Facilities Division, Argonne National Laboratory
9700 S. Cass Ave. Argonne, IL, 60439 USA

ABSTRACT

The Personnel Safety System (PSS) for an experimental beamline at the Advanced Photon Source (APS) uses a dual-chain, PLC-based architecture for controlling shutters and experimental stations. Reporting the status of these two chains is important from both an operational and diagnostic standpoint. This paper discusses the methods used for passing information from both chains of the PSS to the EPICS-based control system at the APS.

I. INTRODUCTION

The design of the Personnel Safety System for the experimental beamlines at the Advanced Photon Source (APS) specifies the use of Programmable Logic Controllers (PLCs) for both interlock and control functions. To meet certain DOE requirements, the PSS was designed using two chains in an effort to avoid common-mode failures [1]. The first chain (denoted "Chain A") uses the Allen-Bradley PLC-5 series processor with both local and distributed I/O. The second chain ("Chain B") uses the GE/Fanuc 90-70 processor, also with both local and distributed I/O. Each beamline has its own autonomous PSS with no knowledge of, or communication with, any other beamline [2].

The (physical) size of the APS makes it necessary to be able to monitor the status of any given beamline from a remote location (such as the control room). To accomplish this without compromising the individuality of each PSS, special hardware and software has been added to both chains. These additions allow each chain to report a variety of information (including I/O points, status information, faults, etc.) to various EPICS database records. Once it is available as a collection of EPICS database records, any authorized X-Terminal or workstation has the ability to display the status of each beamline PSS. This information can be used for a variety of things including validation, debugging, fault tracing, and user training.

II. CONCEPTUAL DESIGN

The overall design requirements specified that the individual chain is responsible for sending the appropriate data to the EPICS database. It is also a requirement that the EPICS interface is not an integral function of the PSS operation (i.e., if the IOC goes down, the PSS must still function). Finally, it is also necessary that the communication be a read–only operation (the database must not have the ability to write into the PLC data space). Each chain has special hardware and software to meet these criteria.

III. CHAIN A - ALLEN-BRADLEY PLC-5

Allen-Bradley produces a VME "scanner" card that can be used to communicate with an Allen-Bradley back plane. This scanner typically connects via shielded twisted pair wire to an Allen-Bradley "Remote I/O Adapter," which sits in an Allen-Bradley crate. It is then possible to perform most of the functions of an Allen-Bradley PLC-5 processor – including reads and writes. Note that the "intelligence" of this system comes solely from the VME scanner – the Remote I/O Adapter is a non-intelligent device. At this point, it would seem that the simplest method of transferring data to EPICS from the PLCs would be to use this VME Scanner to Remote I/O Adapter communication. Two overriding concerns prevent this: firstly, the interlock system must be stand-alone and not dependent on the VME scanner, and secondly, any type of write operation (from the VME to the PLC) must not be allowed as it may compromise the integrity of the interlock program. A data transfer method that only <u>reads</u> from the Interlock System is

* Supported by the U.S. Department of Energy, BES-Materials Sciences, under contract W-31-109-ENG-38

required: a one-way communication that "filters" the data the VME side can access. To facilitate such a communication requires the use of an Allen-Bradley "Direct Communications Module" (DCM). This module was designed to allow a remote PLC processor to access the local processors data space (both read and write) through shielded, twisted pair cable ("blue hose"). The DCM occupies a slot in the same chassis as the local processor. Communication between the local processor and the DCM is done directly through the backplane. The local processor decides what data it wants the DCM to read and/or write. It is this ability that allows us to achieve the desired communication protocol [3].

It was decided to pass all of the input and output bits into EPICS, along with certain PLC status information, and a fault stack containing a time-ordered list of faults. To accomplish this, a small amount of code was added to the PSS logic that collects all of this data into two groups and sends them to the DCM via the Block Transfer construct. The DCM is configured to buffer the blocks (a block can be up to 64 words - 16 bits each) coming in over the (Allen-Bradley) backplane and send them over the blue hose when the VME scanner requests them. A new EPICS record type, denoted the "abDcm," is used to specify such parameters as the scanner address (in "Link, Rack, Slot" notation), number of word groups to read, and scan frequency. For each binary value (mostly input and output bits) a "bi" record is created for use in EPICS applications such as MEDM and the Alarm Handler. In an attempt to reduce network traffic, all binary input records are interrupt scanned [4].

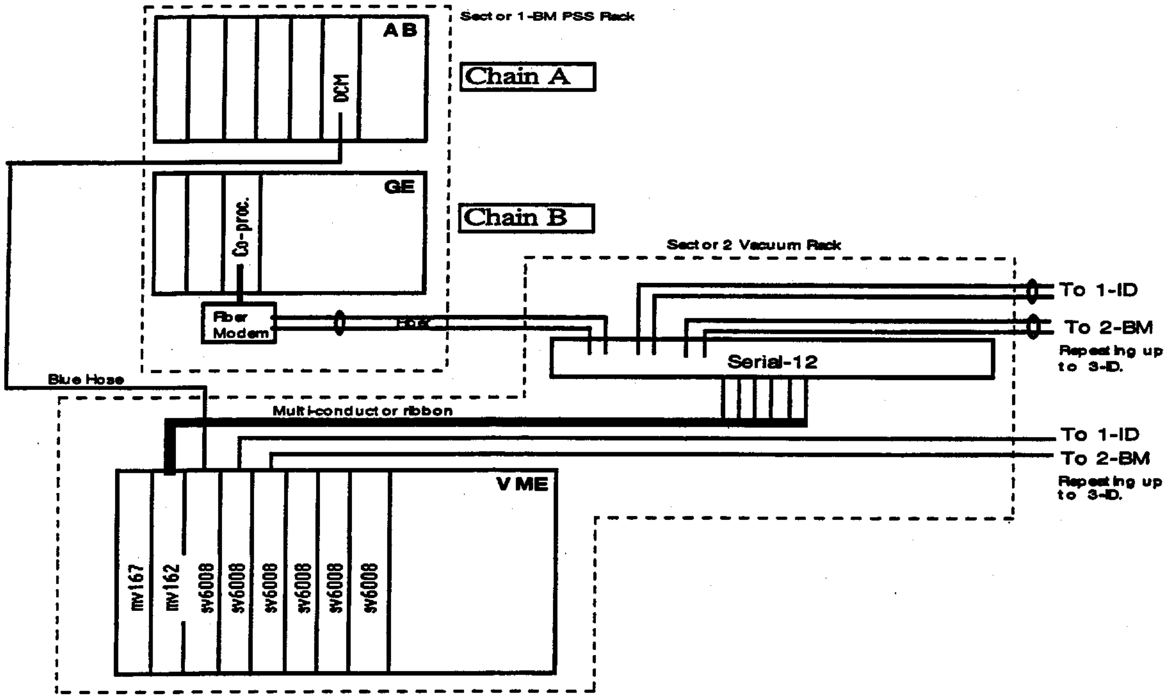


FIGURE 1 - SCHEMATIC LAYOUT OF IOC AT SECTOR 2

Each IOC supports six beamlines (see figure 1). In order to maintain isolation between beamlines, six VME scanners are used (each with a different link number). The scan time for each DCM is fixed at one second, making the best-case resolution for updates about once per second (chain A).

IV. CHAIN B - GE/FANUC

General Electric makes a line of 90-70 coprocessors that sit on the GE backplane and are capable of running programs independently of the main processor. All main processor memory is available for inspection by the coprocessor, including I/O and status. We choose a coprocessor with configurable serial ports that can be programmed in a extended BASIC language, which GE named "MegaBasic". MegaBasic was chosen due to its large flexibility in controlling character output format to the serial ports, along with the relative ease of programming.

A small MegaBasic program was written that continuously scans the input, output, and status bits of the main processor, forms them into a string of hexadecimal characters, and prints them out of the serial port. A single frame consists of a start flag (the ASCII character "s"), the byte representation of the main processor memory, an end flag (letter "t"), and a checksum. Frames are sent continuously with no regard to acknowledgments. The serial port of the coprocessor is connected to a fiber-optic modem to allow long cable runs without (electical) noise problems. The other end of the fiber pair is connected to another fiber modem (one of many concentrated in a "serial 12" chassis - see figure 1), which in turn is connected to the VME module (described below).

On the VME side, a MVME162 module running the Hideos operating system is configured with two quad serial IP (Industry Pack) modules. Each of the IP modules is capable of communicating with four serial (RS-232) devices. The serial port of the GE co-processor is connected to one of the eight available serial ports on the MVME162 via fiber. A Hideos task was written to read the status frames arriving from the coprocessor and check for successful transmission (via the framing bits and the checksum). If a valid transmission is not received in five seconds of the last received frame, the connection is assumed dead and the record support specifies an invalid alarm.

Assuming a successful transmission, the Hideos task then looks at groups of two bytes (a "bit-group") and checks to see if any bit has changed within a group. If so, the associated records within the bit-group are signaled to the process and the values are updated appropriately [5]. As in the Allen-Bradley implementation, a separate binary input record is created to map each individual I/O point of the Chain B PSS.

Monitoring chain B takes only one MVME162 card since the IP modules allow us to connect to six individual GE coprocessors. The communication between the GE coprocessor and the MVME162 takes place at 19.2 Kbps. Assuming a low noise connection (which is valid since we are using fiber as the transmission medium), we will experience very few corrupted frames and can easily match the one second resolution of Chain A.

V. OPERATOR INTERFACE

MEDM was used to build a variety of screens (see figure 2). In general it was desired to be able to monitor all input and output points for both chains simultaneously. A special database was created to compare related outputs between chains and is used to flag abnormal conditions (for example, if an input does not appear on both chains when it is normally supposed to). Other screens include a fault stack (showing the last 10 faults), a PLC status page (showing a variety of PLC–specific information) and navigation (via menus).

Other screens show a schematic overview of the beamline and experimental stations. These screens are more useful to the experimenters to indicate status of shutters, doors, etc. (figure 3).

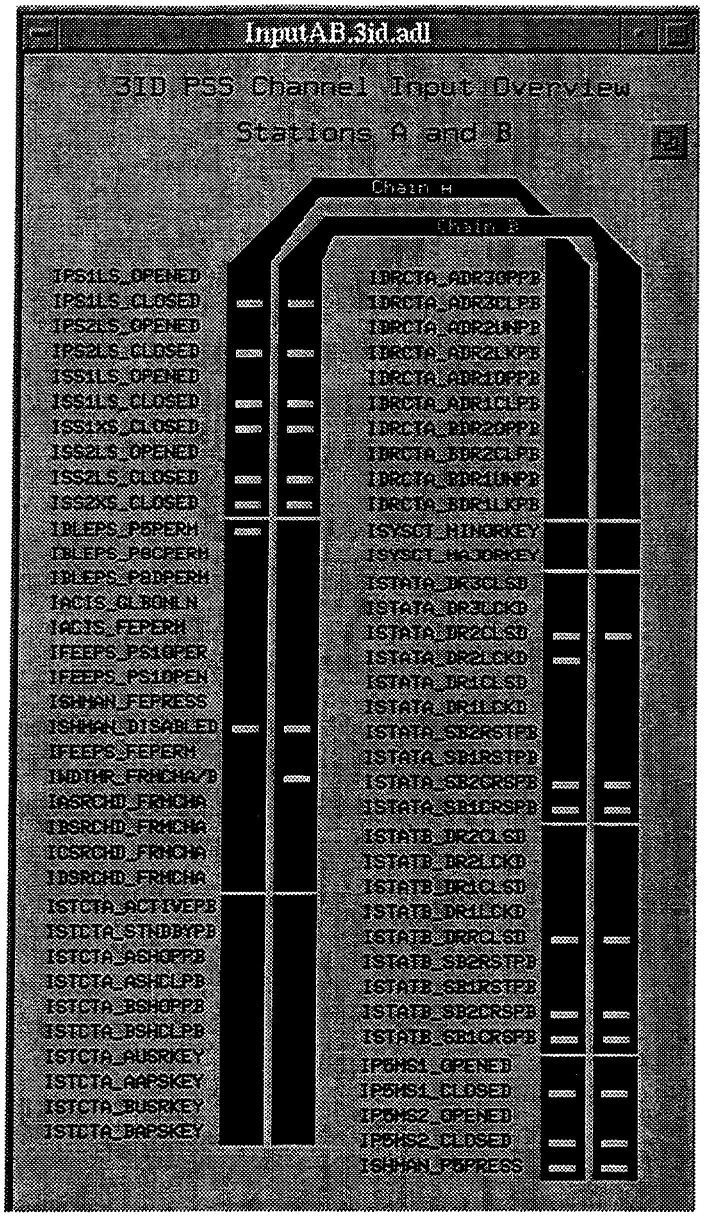
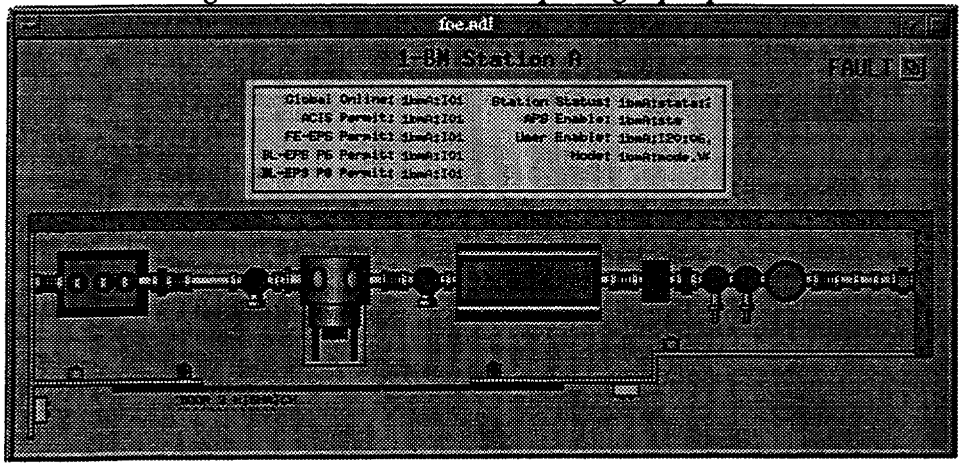


Figure 2 - MEDM Screen Depicting input points



Global Online: 1bmA:I01
ACIS Permit: 1bmA:I01
FE-EPS Permit: 1bmA:I01
BL-EPS P6 Permit: 1bmA:I01
BL-EPS P8 Permit: 1bmA:I01
Station Status: 1bmA:state:
APS Enable: 1bmA:sta
User Enable: 1bmA:I20:06,
Mode: 1bmA:mode.W

Figure 3 - MEDM Screen depicting a schematic view of an experimental station

VI. CONCLUSIONS AND POSSIBLE IMPROVEMENTS

Currently the system is implemented for one beamline (3-ID) and is working very well. All of the diagnostic screens have been used for beamline hardware checkout and as an aid to software verification. The ability to monitor the entire I/O map of both chains simultaneously has been a tremendous benefit to diagnosing wiring problems and limit–switch misalignments.

A number of improvements and/or additions can be added to the system however. The interrupt driven nature of both chains is worthwhile in theory, but may not do so well in practice. There are a few signals that are periodically changing on the I/O maps which need not generate interrupts (particularly the "watchdog" signal – a pulse train traded between both chains to indicate "I'm alive" status [2]). Making special consideration for these periodically changing signals may cut down on network traffic and processor load.

At this point in time, Chain B does not pass any status information (i.e., scan times, battery condition, etc.). Although it was deemed not necessary due to the simple interlocking function of that chain [2], it will be added later to assist in debug and fault tracing.

Finally, the Alarm Handler and the Archiver have not been implemented up to this point. It would be worthwhile to log certain I/O and status points in order to reconstruct any run time faults that may occur during beamline operation.

VII. ACKNOWLEDGMENTS

I would like to thank Jim Kowalkowski for the tremendous work done on Hideos in general and the Fanuc support in particular. I would also like to thank Chris Seaver for assistance in all questions regarding the Allen-Bradley PLC systems.

VIII. REFERENCES.

1. DOE Order 5480.25, "Safety of Accelerator Systems," (DOE 1992b).

2. Hawkins, J., et al., "Personnel Safety System for the Beamlines at the Advanced Photon Source," SRI Conference proceedings, 1995.

3. Stein, J. "One-Way Data Transfer for PLC to VME Status Reporting at the Advanced Photon Source," Nuclear Instruments & Methods in Physics Research - A 253 (1994) 210-212.

4. Stein, J., Nawrocki, G. and Kramer, M. "Data Exchange from Allen-Bradley PLC Based Systems to EPICS," ICALEPCS Proceedings, 1995.

5. Kowalkowski, J. "A Cost Effective Way to Operate Instrumentation Using the Motorola MVME162 Industry Pack Bus and HiDEOS," ICALEPCS Proceedings, 1995.

Data Exchange from Allen–Bradley PLC–Based Systems to EPICS at the Advanced Photon Source*

S. J. Stein, C.L. Seaver
Experimental Facilities Division

G. J. Nawrocki, M.R. Kraimer
Accelerator Systems Division

Argonne National Laboratory
9700 S. Cass Ave. Argonne, IL, 60439 USA

ABSTRACT

The Advanced Photon Source (APS) uses Allen–Bradley Programmable Logic Controllers (PLCs) in a number of applications including equipment protection and personnel safety systems. It is necessary to report the operating status of these systems to the main control system, designed using the EPICS tool kit. This paper discusses the method used at the APS to transfer data from a PLC–based subsystem to the EPICS–based control system.

I. INTRODUCTION

Traditionally an EPICS database communicates with the outside world via a number of interfaces - one being the Allen–Bradley Remote I/O link. This link consists of an Allen–Bradley VME scanner card (SV-6008) that resides in a slot in the Input–Output Controller (IOC) and is connected via "Blue Hose" (twinax cable) to a "Remote I/O Adapter". The Remote I/O Adapter resides in slot zero of the Allen–Bradley 1771 backplane and functions as a bus arbiter but contains no user code. All of the intelligence is contained within the IOC (usually a MVME167 processor running an EPICS database). With proper configuration, an EPICS database can access a variety of Allen–Bradley 1771 I/O cards including binary, analog and a few special cards (thermocouple, strain gauge, etc.). This method was used for many years as it was a cost-effective way of adding many remote I/O points to an IOC with little effort.

Within the last few years, however, it was determined that another method of acquiring data from an Allen–Bradley system was needed – a passive method that would grant an EPICS database the ability to view the status and I/O map of an autonomous PLC–based system. This was born from the need to monitor the status of two large safety systems at the Advanced Photon Source (APS) which use PLC processors running ladder logic (a graphical–based control language). Since the PLC processor resides in slot zero of the Allen–Bradley backplane, the remote I/O adapter is not an option. This paper discusses the hardware and software used to communicate information from a PLC control/interlock system into EPICS.

II. OVERVIEW

As mentioned in the introduction, the standard connection from an IOC to a Remote I/O adapter is not valid. Instead, the use of the Allen–Bradley Direct Communication Module (DCM) is appropriate. The DCM is meant to be used as a method of data transfer from one PLC system to another. The DCM resides in any open slot in the Allen–Bradley crate, and the main processor controls reading and writing to it via the backplane. Typically, the DCM is connected to another processor (in a different system) using blue hose. The remote processor believes it is talking to a remote I/O adapter - the communication over the blue hose takes place using the same protocol as the Remote I/O link.

If we replace the remote processor with the VME scanner, we now have the ability to read the status of a PLC system via an EPICS database. Previously it was necessary to "fool" the EPICS device support into reading from the DCM as no specific code was written to talk to the DCM [1]. With the latest

* Supported by the U.S. Department of Energy, BES-Materials Sciences, under contract W-31-109-ENG-38

revision of EPICS however, a specific hybrid record type was included for configuring and communicating with the DCM.

III. EPICS PROGRAMMING

As stated above, the latest versions of EPICS support communication with the DCM via some special record types and device support. A new record type ("abDcm") was created that is used to specify such items as the Link, Rack and Slot address of the DCM, the scan rate and the number of messages that are to be sent from the DCM (1 message = 64 short words) (see figure 1). The Allen–Bradley driver (drvAb) was modified slightly to allow for arbitrary block transfer requests (instead of tying the block transfer to specific I/O modules). Processing of the abDcm record causes a read from the DCM - if a word from any of the messages has changed, a Channel Access monitor is triggered. Typically records are created as INST_IO pointing to the abDcm record and specifying the message number, word number, and bit number (if needed). An example bi (binary input) record is given below (figure 2):

```
record(abDcm, "TestDCM:SJS")
{
        field(SCAN, "1 second")
        field(UT0, "YES")
        field(UT1, "YES")
        field(LINK, "x")
        field(RACK, "y")
        field(SLOT, "z")
}
```

- The first field specifies that this record be scanned periodically - once per second. Note that the scan rate can be increased or decreased depending on the resolution desired versus the processor load that can be tolerated.
- The second and third fields specify that the DCM will be sending two messages (there are actually six of these fields UT0 through UT5 - if the UT field is not specified, it is assumed to be "NO").
- The last three fields specify the address of the DCM in Allen–Bradley "Link, Rack and Slot" notation.

Figure 1 - Example of abDcm record

```
record(bi, "TestBi:SJS")
{
        field(SCAN, "I/O Intr")
        field(DTYP, "Ab Dcm")
        field(INP , "@TestDCM:SJS.Tx[y,z]")
}
```

- The first field specifies that this binary input record be I/O Interrupt scanned - this record will only process if the word (specified in the INP field) changes.
- The second field specifies that the device type is the AB DCM.
- The last field points to the record name of the DCM record and specifies the message number (the "x"), the word number (the "y") and the bit number ("z"). Note that specifying the bit position within the word is valid only for a binary input record, other record types (namely mbbi and longin) would only specify down to the word number.

Figure 2 - Example of binary input record

The abDcm device support expects to receive at least two messages per data transfer. Each message may consist of up to 64 short words (16 bits), however the first eight words are reserved as described below (see figure 3). The first word is used by the DCM to send status information including buffer state, last transmission success, etc. The second and third words are reserved for future use and the fourth word (word 3) contains an integer value corresponding to the message number (this is the number specified in the Tx parameter in the Param field of the database record). The last four words are also reserved for future expansion. Note that this configuration makes the ninth word the first one available for user data.

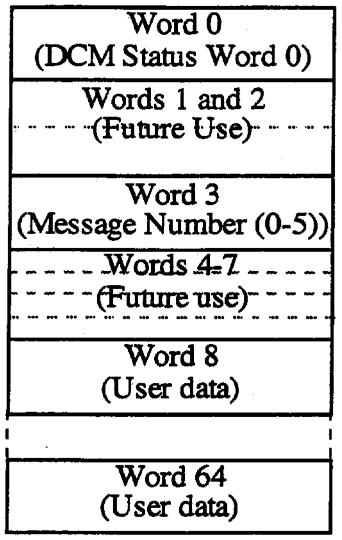


Figure 3 - Message structure from DCM to EPICS

IV. ALLEN–BRADLEY PROGRAMMING

The Allen–Bradley PLC system must be modified slightly to add the monitoring capability. Firstly, the messages are constructed in a certain sequence that is known to both the EPICS and PLC programmer. The messages are then transferred to the DCM via the Block Transfer statement within the ladder logic program. Each message is identified by a unique descriptor (an integer value) which is placed in the fourth word. Each different message is sent in sequence, one at a time. After a message is received in the DCM, block transfers into the DCM are blocked until the message is transmitted over the blue hose. This process is continuous.

V. SYSTEMS IN USE

A variety of subsystems of the APS use the Allen–Bradley series 5 processors for control and interlock functions. In general it is important to be able to monitor the status of these systems from many locations including the Main Control Room (MCR). Below is a short description of three currently implemented monitoring packages at the APS.

V.1 ACIS

The purpose of the Access Control Interlock System (ACIS) is to guarantee that no one is in the APS accelerator tunnels while the accelerator is in operation. Since the APS is composed of several accelerators, there are actually four separate ACIS systems. This allows portions of the APS to be in operation while people have access to other parts. Each ACIS system consists of two redundant Allen–Bradley PLC–5 systems.

EPICS currently is used to display the status of two of the ACIS systems with plans to include the other two systems. For the two systems currently monitored, the EPICS database consists of approximately 1700 records, most of which are binary input records.

In building the EPICS database and the MEDM (an EPICS GUI) displays, extensive use was made of the macro expansion capabilities of dbLoadTemplates and MEDM. The 1700 records are generated from just a couple hundred record templates. Over 100 MEDM displays are available but are generated from just a few medm template displays.

Although the EPICS database has > 1700 records, the CPU utilization is less than 5%. This is because the DCM support provides the ability to process records only when the word containing the records signal changes.

V.2 PSS

The Personnel Safety System (PSS) for the beamlines at the APS is used to provide access interlock and control along an experimental beamline. Each system consists of two independent chains – one of which is based on the Allen–Bradley series 5 processor.

Currently the PSS for the Sector 3 Insertion–Device beamline has a completely functional PSS monitoring package installed. It has been used for a variety of tasks including initial software and hardware debugging, software and hardware verification, on-line diagnostics and fault tracing. At present, the Allen–Bradley chain of the PSS sends two messages at each transfer. Each message consists of a variety of values including the entire input and output image tables, PLC status information (including the current time and date of the PLC clock) and a time–ordered stack showing the first ten PSS faults [1].

A database was constructed that consists almost entirely of binary input records (see figure 1) corresponding to the input and output table of the PLC. Macro substitution was used (via the dbLoadTemplates command) due to the extremely consistent nature of the I/O map. In general each bi record is named in a method that correlates easily to the Allen–Bradley addressing scheme.

Another database was constructed that is used to store "static" records that do not fit into the macro substitution mold. These include a few longin records used to pass time and date, ten records that correspond to a fault stack, and a variety of "analog" PLC status values (scan time, battery status, etc.)

Finally a third database is used solely to service the DCM. It contains one abDcm record (see figure 2) that is periodically scanned once per second [2].

V.3 METASYS

Metasys is a distributed system of intelligent I/O processors used for control and monitoring of environmental and HVAC parameters at the APS. The Metasys Network port is a device that allows monitoring and controlling of the Metasys Facilities Management System from a third party computer system. The Metasys Network Port emulates the communication of a PLC-5 and communicates with other PLC-5 processors via RS232 using the Allen–Bradley Full-Duplex Data Link Layer Protocol (DF/1). An Allen–Bradley 1771 KF2 converts the Network Port RS232 signal to a Data Highway Peer Communication Interface to complete the Network Port to PLC-5 link. This dedicated PLC-5 module is then programmed to act as bridge between the Metasys Network port and EPICS utilizing the DCM device and record support.

Currently, at the APS, a small set of binary and analog Metasys values are passed to EPICS as a proof of concept. This number will increase as more environmental data is incorporated into EPICS displays, alarm handling applications, and data acquisition techniques familiar to the operators and accelerator physicists [3].

VI. CONCLUSION

The additions and modifications to allow communication via the DCM have proven to be invaluable for monitoring PLC–based systems at the APS. The ability to view the I/O and status of these systems has made debugging, diagnostics and fault tracing much easier than previous methods.

Currently there are three systems in place at the APS which use the DCM as a bridge into EPICS. The Accelerator Control Interlock System (ACIS) uses a very large number of records to convey the status of a large system to operators throughout the ring. The Personnel Safety System (PSS) also consists of a large number of records used to report status to operators and, in the near future, APS users. Finally, the Metasys system also reports (indirectly) through a DCM to the main control system a variety of environmental system parameters.

VII. REFERENCES

1. Stein, J. "One-Way Data Transfer for PLC to VME Status Reporting at the Advanced Photon Source," Nuclear Instruments & Methods in Physics Research - A 253 (1994) 210-212.

2. Stein, J. "Personnel Safety System Status Reporting via EPICS at the Advanced Photon Source," ICALEPCS Proceedings, 1995.

3. Nawrocki, G. Seaver, C. and Kowalkowski, J., "Monitoring Commercial Conventional Facilities Control with the APS Control System – The Metasys to EPICS Interface," ICALEPCS Proceedings, 1995.

Magnet Power Supply Controls of the SPring-8 Storage Ring

H. Takebe, T. Fukui, K. Kumagai, T. Masuda, J. Ohnishi, A. Taketani, R. Tanaka, T. Wada, W. Xu and A. Yamashita.

SPring-8, Kamigori, Ako-gun, Hyogo, 678-12, JAPAN

Abstract

SPring-8 Storage Ring (SR) magnet power supplies (PS) were installed in the PS-rooms in March and October of 1995. High output current cables for the large DC PSs were put on the trays above the machine tunnel. A large number of small size power supplies are now under construction. A fiber distributed remote I/O (RIO) slave card was installed into the PS unit and connected to the RIO master card in VME. A test program for the power supply system using an RIO device-driver was developed on an HP-RT workstation [1]. Magnet PS control programs are now in being written.

1. Introduction

The SPring-8 storage ring consists of 48 Chasman-Green type cells. Each cell has two Bending magnets (B), ten Quadrupole magnets (Q), seven Sextupole magnets (S) and twelve Steering magnets (St). Forty auxiliary PSs (QA) for the long straight section cells are adapted for the Quadrupole magnets [2]. This number will come to 480 in the future [3]. Thirty-eight sets of Skew Q and eight injection magnets are to be installed. The total number of Steering magnet PSs (St-PS) is 576. Thus the total number of the SR magnet power supplies is 1120. The 480 QA supplies must electrically float several hundred volts above ground level. Due to considerations of isolation, noise rejection and reliability, an optical fiber linked Remote I/O system was chosen. A large number of beam position monitoring circuits plus vacuum control also use this RIO system, which consists of the following devices: 1) VME master with a dual port memory, 2) RIO (slave card), of which there are seven different types, 3) star branch for optical fiber cables from the master module to the slave cards [4][5].

2. Magnet Power Supplies

The total numbers of large PSs for B, Q and Sx is 18. They were installed in PS Room-A in March 1995, and all St-PSs on 6 October 1995. They are thyrister controlled (B: 24th Q: 12th, S: 6th phase) DC supplies [6].

The steering magnets have independent power supplies (St-PS). One hundred ninety-two sets of St-PS and 10 sets of QA are located in four PS rooms. Q magnets, which are connected in series, are adjusted by QA to correct the modulation of the beta function and phase advance [2][7]. St-PS and QA are switching regulator type. Forty sets of QAs and 576 sets of St-PS were also completed and installed.

A Four Bump, one DC-septum and one Pulse-septum magnet PSs (SR injection devices) are designed and under construction. The Bump and Pulse-septum PSs use RIO type-A control circuits (see figure 1). These also use a timing system connected to the 508.58 MHz master oscillator. The DC-septum PS takes RIO type-B.

3. PS Control System

Each reference voltage for the large (B, Q, S) PSs is given by a 16 bit DAC controlled by the digital output of an RIO type-B. These DACs for the B, Q, and S PSs are installed in a temperature controlled box of the PS cubicle. This RIO card has 32 bit DI and DO. Figure 1 shows the BP, QP and SP in PS Room-A.

The St-PS, QA and QB are controlled by the analog output (-10 ~+10 V) of an RIO type-A. The RIO type-A is attached to that PS chassis with the same guard level. The RIO operational power source (5V, ±15V) is supplied by the floated PSs. The RIO type-A also has a double integration ADC, which monitors an actual current using a shunt resistor output (-1 ~ +1V). These PS units have 8 bits of status (power on/off, DC-bus on/off, remote/local, fault, over-current, temperature, ext-interlocks, etc.) to be read.

The currents of the multiple PSs must be changed simultaneously for an orbit correction. Therefore the orbit correction program requires any combination of four or more St-PSs in any PS rooms. Also a slow (rough) timing system (~ 50 ms) is needed. The computer network is used for such rough timing.

4. VME-RIO Network and PS Interlock System

All the RIO slave cards for all the magnet PSs are controlled by only four VME crates by using optical branch cards. The SPring-8 control system adopted FDDI and Ethernet to communicate between WSs and VME crates. The SR magnet PS setup and tuning will also be done through this FDDI.

For a closed orbit correction, multiple sets of steering magnet PSs must be set simultaneously (local bump orbit). Thus a rough timing system (~50 ms) must be installed for magnet control. The RIO's master to master [8] communication (data transfer time is less than 0.8 ms) system can be used if the optical fiber cables are distributed between the four PS rooms as an RIO network [4]. Furthermore if RPC response time is less than 50 ms the FDDI network can be used. (In cases using TCP/IP or UDP the mean response time of RPCs with the HP-RT was measured to be a few ms).

A programmable logic controller (PLC) is used for the PS interlock system. The water flow switches and thermostat switches of all magnets are connected as distributed input units of the PLC.

Figure 1 shows the VME and RIO network and an interlock system for the magnet PSs. The water flow switches and coil thermostat switches are connected to the PLC (Sequencer) remote units, which are located beside the magnet bases, using twisted pair cables. A normal relay close signal from the PLC output card is connected to the PS's interlock input. Primary alarm information goes to a VME from a PLC's Ethernet port. Five PLC controllers are connected through an optical fiber cable in the maintenance corridor.

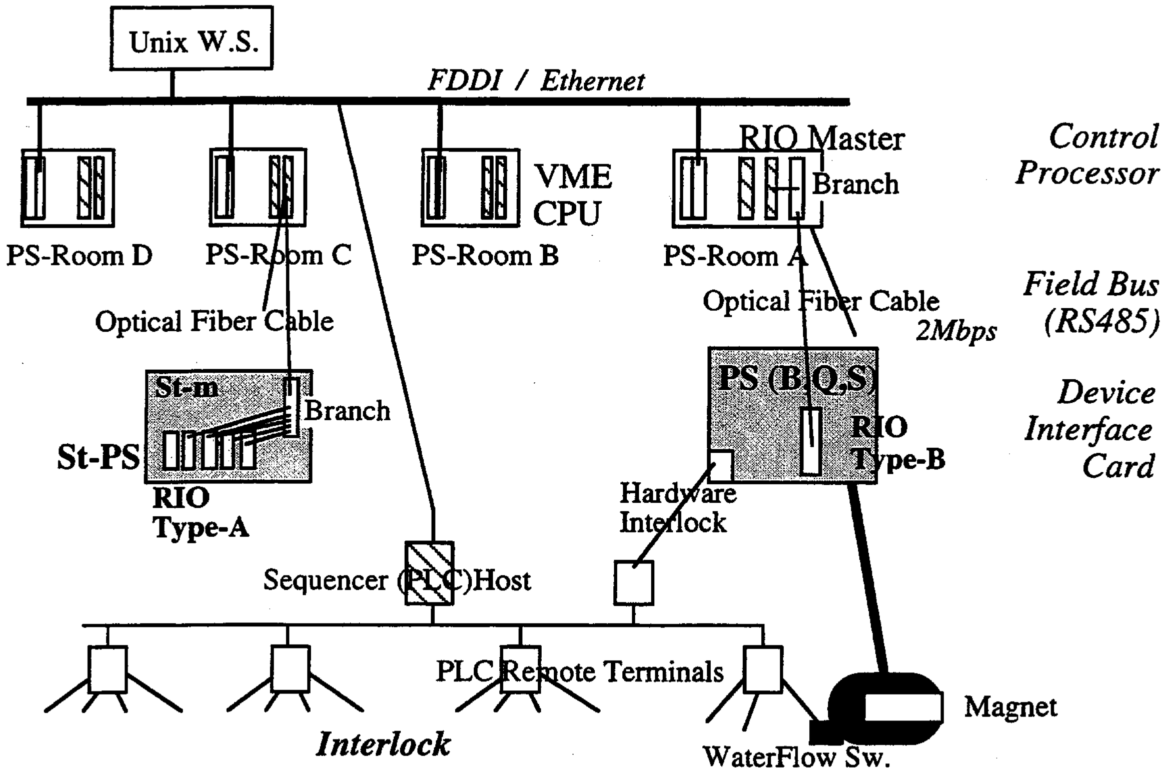
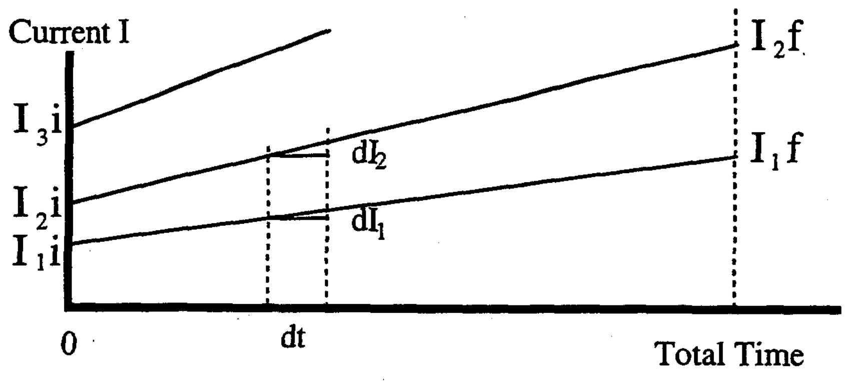


Figure 1. VME-RIO network and PLC (Sequencer) interlock system for magnet PSs.

5. Software

A test program of the VME computer was written for the HP-RT, and a test operation was made with an St-PS for on/off, reset and current up/down control. For the multiple current set sequence of the St-PS, messages from a Message Server (MS) to an Equipment Manager (EM) have been designed [9]. Magnet name, final currents and set times are sent to the EM from an upper layer program. Then the EM gets initial currents from the database table and calculates the step currents and step times (dI and dt), using the current value limitations (see figure 2).

The database structure and the rough timing system of the magnet PS are now being studied. A GUI program and man-machine interface programs will be completed by summer 1996. All installation of the optical fibers into the 576 PSs will be done in December 1995. High power operation for the Q PSs will be done with the 48 Q-magnets connected in series after the SR ring vacuum test in April 1996.



MS --> EM Send Data: Magnet Name, I final, Total Time

EM: $dI/dt < (dI/dt)$ max

Figure 2. Multiple current setings must be made simultaneously. The Equipment Manager (EM) calculates the step currents (dI) and step times (dt) and checks the maximum dI/dt.

6. References

[1] R.Tanaka et. al., this Conference.
[2] H.Tanaka et. al., RIKEN Accel. Progr. Rep. vol.24, 1990, p141.
[3] H.Takebe et. al., RIKEN Accel. Progr.Rep., vol.24, 1990, p156.
[4] H.Takebe et. al., Proc. of EPAC'94, London, 1994, p1827 .
[5] S.Harada, K.Matsuo, and M.Hasegawa. Tech. Rep. of Mitsubishi Elec.Corp., "Multi-drop system", Mitsubishi El. Co., Nishi-Gotanda 2-21-1, Shinagawa-ku, Tokyo 141, JAPAN.
[6] H.Takebe, et.al., RIKEN Accel. Progr.Rep., vol.27, 1993, p147.
[7] H.Tanaka et. al., RIKEN Accel. Progr. Rep. vol.25, 1991, p184.
[8] H.Takebe, et.al., RIKEN Accel. Progr.Rep., vol.26, 1992, p176.
[9] A.Taketani et al., this Conference.

Fast Bunch Integrator
A System for Measuring Bunch Intensities in the Fermilab Main Ring

Jeff Utterback, Greg Vogel
Fermi National Accelerator Laboratory*
POB 500, MS 307, Batavia, IL 60510 USA
Paper Prepared for ICALEPCS 95

ABSTRACT

In order to support the proposed luminosity increases in the Fermilab Tevatron Collider, a new Fast Bunch Integrator (FBI) system has been developed. FBI is used to precisely measure the intensity of each bunch in the Main Ring before injection into the Tevatron. These intensity measurements are particularly useful for monitoring the coalescing process as it takes place. This newly designed FBI system will also be used in the Main Injector when complete. The FBI system is comprised of a VME Front End which communicates with Fermilab's ACNET via an Ethernet connection. The boards in the VME crate are a Motorola 162 CPU, a Fermilab designed Pulse Pattern Generator (PPG), 2 Fermilab designed Integrator boards (one for protons and one for antiprotons), and 2 "Comet Analog Digitizer" boards by Omnibyte corporation. The embedded software makes use of the VxWorks operating system by Wind River Systems.

Precise timing pulses are produced by the PPG to trigger the integrators and the digitizers just as the bunches are passing the detection equipment in the beam pipe. The embedded software is required to program the PPG so that the pulses occur at the correct times. The embedded software also has real-time requirements because the intensity readings must be retrieved from the hardware quickly to make the data available to ACNET for console displays and for continuous plots.

Software has been created to produce a local terminal display which mimics the remote console access. This local display is useful for system access in the field and for debugging purposes.

INTRODUCTION

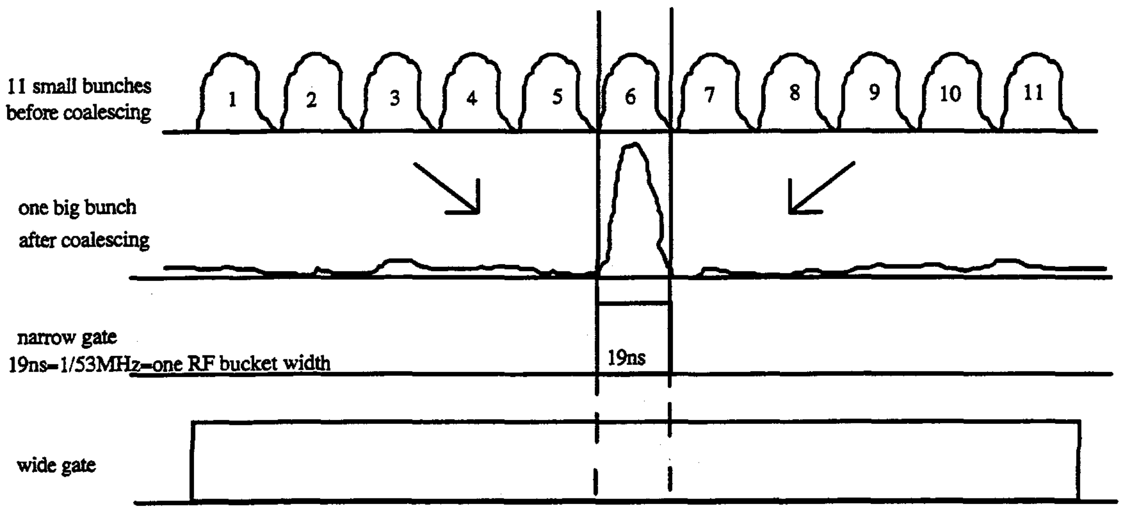
The Fast Bunch Integrator (FBI) is a system which monitors the bunch intensity of the protons and antiprotons in the Main Ring (or the future Main Injector) and supplies this data to ACNET. The data will be particularly useful during the coalescing procedure. The data is supplied in a suitable form for continuous Fast Time Plotting (FTP) and Snap Shot Plotting (SNP). The data is also supplied in a suitable form for all normal slow ACNET accesses such as Parameter Pages, and Data Loggers.

THEORY

In order to increase the luminosity of colliding beam physics in the Tevatron, a process of coalescing must take place in the Main Ring before injection. This process packs the Main Ring batches into high intensity bunches. The end result of the coalescing process is 12 high-intensity proton bunches and 4 high-intensity antiproton bunches. In order to get 36 on 36 in the Tevatron, 3 injections of 12 proton bunches and 9 injections of 4 antiproton bunches will be required.

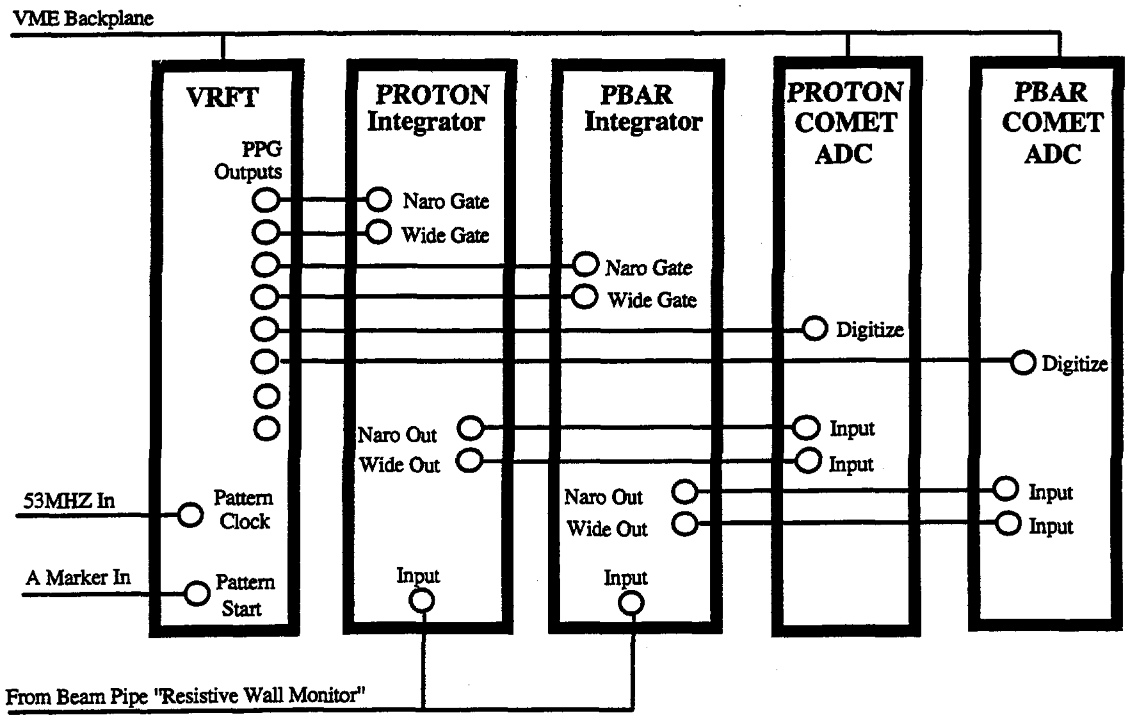
The goal of FBI is to get an intensity reading for each coalesced bunch in the Main Ring. We are not trying to create a "scope picture" of the bunches, we are simply trying to get an intensity value for each bunch. To achieve this goal, the intensity signal is used as an input to an integrator circuit. The output of the integrator depends upon the magnitude of its input and upon the amount of time it is allowed to integrate. The integrator takes two readings on each bunch; a wide gate reading and a narrow gate reading. The wide gate reading is used to determine the intensity of the batches before coalescing. The narrow gate reading is used to determine the intensity of the bunch after coalescing. The narrow gate reading should go higher and higher as the coalescing process takes place. The wide gate reading should not change much during the coalescing process but may decrease slightly due to losses incurred by the coalescing process.

*Operated by the Universities Research Association under contract with the U.S. Department of Energy.


11 small bunches before coalescing
1
2
3
4
5
6
7
8
9
10
11
one big bunch after coalescing
narrow gate 19ns=1/53MHz=one RF bucket width
19ns
wide gate

HARDWARE

The FBI project consists of a VME crate connected to ACNET via Ethernet. The boards in the crate are a Motorola 162 CPU, a Fermilab-designed VUCD Tevatron clock decoder board, a Fermilab-designed VRFT pulse pattern generator board, 2 Fermilab-designed integrator boards, and 2 Comet digitizer boards from Omnibyte Corporation.



THE VRFT PULSE PATTERN GENERATOR

The VRFT pulse pattern generator (PPG) is used for all of the critical timing. It has 8 outputs which are driven by the pattern generator. The pattern can be programmed so that pulses of various widths occurring at various delays from the "Pattern Start" can occur on any output. The pattern memory is 64K bytes long so a maximum pattern length of 64K buckets can be used. Each bit in a pattern byte corresponds to one of the 8 outputs. The pattern proceeds from one byte to the next as the 53MHz clock ticks occur.

The PPG outputs are used for the integrator gate pulses as well as the trigger pulses for the digitizer boards. The PPG is triggered to start a pattern on the Main Ring A Marker. This signal occurs once per turn. The PPG clock runs at 53MHz so the minimum output pulse width is 1/53MHz=19ns=1RF Bucket. This makes it very convenient for the experts to set the pulse delays in terms of "Buckets off the A Marker."

THE INTEGRATOR BOARDS

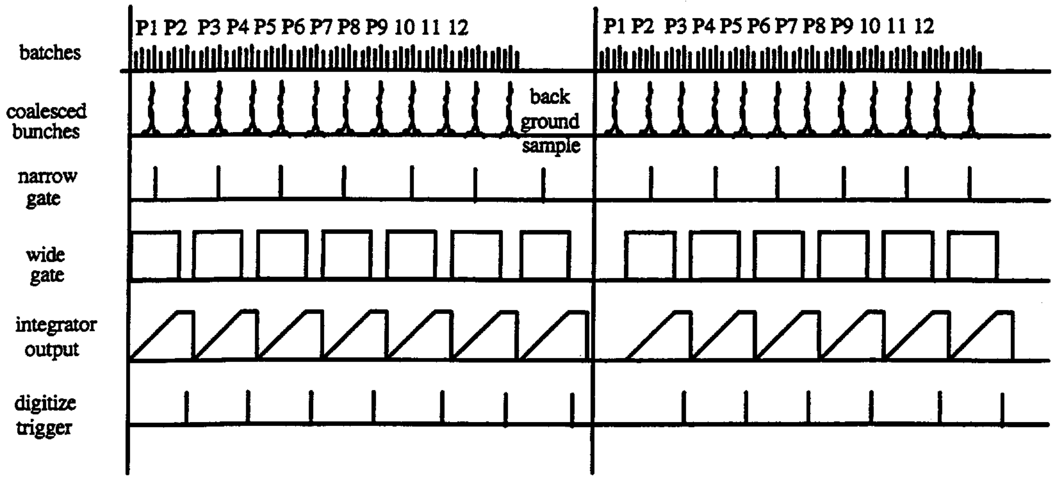
The Integrator boards were designed at Fermilab. These boards take their input from the "Resistive Wall Monitor" in the Main Ring beam pipe. The purpose of these boards is to integrate the input signal upon command from the gate pulses that are provided by the PPG. The narrow gate is programmed to occur just as the coalesced bunch comes by. The wide gate is centered upon the narrow gate and is used to integrate several pre-coalesced bunches. The output of the integrator boards provides intensity signals to the Comet digitizer boards.

In order to get accurate intensity readings, it is necessary to periodically take a reading when no bunches are passing by in the beam pipe. This is called a "background sample". The background sample must be subtracted from all of the bunch intensity readings. It was originally thought that the background subtraction would be done in the Integrator boards, but the design has proven to be too difficult and now the background subtraction is done in software.

THE COMET DIGITIZERS

The Comet digitizer boards contain a large amount of memory. Each time a digitize trigger occurs it puts the new data in the next consecutive memory location. When the PPG pattern takes place, the data items stack up in the Comet memory until the pattern ends and then the microprocessor can retrieve the data.

The time between bunches after coalescing is 378ns. We sample every other bunch at 756ns. In order to get readings on all bunches, we take readings over 2 turns. The first turn yields the odd numbered bunches and the 2nd turn yields the even numbered bunches. The PPG gets its "start pattern" signal from the Main Ring A Marker. The A Marker occurs once per turn and is always present when the Main Ring is running.



A Marker Turn 1
RFT begins Pattern Generation

A Marker Turn 2
RFT ignores this one and continues pattern

The PPG is programmed so that its pattern runs for 2 turns. The PPG will ignore the 2nd A Marker because it is designed to ignore triggers until the pattern is complete. After 2 turns have elapsed, one reading for all 12 Proton bunches and 4 Pbar bunches will be sitting in the respective Comet boards memories. The readings will be in interleaved order such as P1 P3 P5 P7 P9 P11 P2 P4 P6 P8 P10 P12 in the Proton Comet and A1 A3 A2 A4 in the Pbar Comet.

SOFTWARE

The embedded software makes use of the VxWorks operating system by Wind River. This is a multitasking operating system that has very good network capabilities. The software makes its data available to ACNET using an Ethernet connection. The embedded software has 3 main jobs to do. It must load the PPG to create the correct pulse pattern for system operation, it must retrieve the intensities from the Comets in a timely fashion and it must make the data available to ACNET for plots and parameter pages.

LOADING THE PULSE PATTERN

From ACNET the experts have SETTING capability for the various pulse delays and pulse widths. Here is an example of the settings.

P1 gate delay 0 buckets	A1 gate delay 30 buckets
P2 gate delay 20 buckets	A2 gate delay 50 buckets
P3 gate delay 40 buckets	A3 gate delay 70 buckets
P4 gate delay 60 buckets	A4 gate delay 90 buckets
P5 gate delay 80 buckets	Pbar background 300 buckets
P6 gate delay 100 buckets	
P7 gate delay 120 buckets	Proton Naro Gate Width 1 buckets
P8 gate delay 140 buckets	Proton Wide Gate Width 11 buckets
P9 gate delay 160 buckets	Pbar Naro Gate Width 1 buckets
P10 gate delay 180 bucket	Pbar Wide Gate Width 11 buckets
P11 gate delay 200 buckets	Background Sample Width 5 buckets
P12 gate delay 220 buckets	Digitize Pulse Delay Off Wide Gate 0 buckets
Proton background 300 buckets	Digitize Pulse Width 1 buckets

These settings are kept in battery backed RAM on the CPU board, so that reasonable settings will always be available after a reboot. Any time a setting is changed from ACNET, the embedded software saves the new setting in BBRAM and then loads the new pulse pattern into the PPG. That way the pattern is all set to go when ACNET asks for intensity data. The pattern settings are not expected to be changed very often.

RETRIEVING THE INTENSITY DATA

When ACNET asks for intensity data a routine is called which performs the following:

1. Enable the PPG external trigger so that the next A marker triggers the pattern.
2. Wait for the pattern to end by watching the Comet data counter.
3. Disable the PPG.
4. Retrieve the data.
5. Sort out the data.
6. Subtract off the background.
7. Return one pattern's worth of intensity readings to the caller.

This routine takes about 250 uS to execute which is fast enough even if it is called at 720Hz for Fast Time Plots. It returns intensity readings for all 12 proton bunches and all 4 antiproton bunches even if ACNET is only asking for one reading. It would be inefficient to attempt to change the PPG pattern to get just the particular bunch that ACNET is asking for. The software just takes the intensity reading that it needs and returns that single reading.

When this project was first conceived, we thought that we would let the PPG board continually play its pattern and let the data stack up in the Comet boards memory to be periodically retrieved by the microprocessor. In that way, data for every turn would be taken and would be available for ACNET. It turns out that it takes 1 uS per word to retrieve data from the Comet which is typical for off board VME access, so retrieving large blocks of data requires a lot of dead time when the PPG must be disabled and several turn's worth of data must be allowed to pass by. Experience has shown that it is better to wait until ACNET asks for data and then enable the PPG for one pattern each time data is asked for. For Snap Shot Plots, the PPG will be enabled for several patterns so that several consecutive turns' worth of data is taken.

USER INTERFACE

The FBI system can be accessed via a normal Fermilab ACNET console parameter page. Bunch intensities are displayed in text form and updated at a 1Hz rate. Bunch intensities can also be display graphically using the standard Fast Time Plot and Snap Shot Plot facilities. The Fast Time Plot updates at a 720Hz rate and the Snap Shot Plot gathers its plot data every other turn. The gate delays and gate widths can be set by a user at the parameter page.

A local terminal display has also been created for the FBI project. The local terminal is simply a standard VT220 terminal attached to the CPU board in the VME crate. Software has been created to display all of the pertinent readings and settings. The local terminal display has proven to be quite useful during the setup and debugging phase.

CONCLUSION

The FBI project is a blend of commercial hardware with custom hardware and a commercial operating system with custom software. The information from the FBI system is made available to the Fermilab ACNET system so that it can be displayed and analyzed using a typical ACNET Console. The end result is a successfully functioning system which meets the real-time needs for information about the bunch intensities in the Fermilab Main Ring.

Drift-Chamber Gas System Controls Development for the CEBAF Large Acceptance Spectrometer

M. F. Vineyard, T. J. Carroll, and M. N. Lack
Department of Physics
University of Richmond, VA 23173

ABSTRACT

The CEBAF Large Acceptance Spectrometer (CLAS) is a superconducting toroidal magnet with a large volume of drift chambers for charged particle tracking. The performance of these chambers depends on accurate monitoring and control of the mixture, flow rate, pressure, temperature, and contaminant levels of the gas. To meet these requirements, a control system is being developed with EPICS. The interface hardware consists of VME ADCs and three RS-232 low-level hardware controllers. The RS-232 instruments include MKS 647A mass flow controllers to control and monitor the gas mixture and flow, MKS 146B pressure gauge controllers to measure pressures, and a Panametrics hygrometer to monitor temperatures and the concentrations of oxygen, water vapor, and ethane. Many of the parameters are available as analog signals which will be monitored with XYCOM VME analog input cards and configured for alarms and data logging. The RS-232 interfaces will be used for remote control of the hardware and verification of the analog readings. Information will be passed quickly and efficiently to and from the user through a graphical user interface. A discussion of the requirements and design of the system is presented.

INTRODUCTION

The primary instrument in Hall B at the Continuous Electron Beam Accelerator Facility (CEBAF) is the CEBAF Large Acceptance Spectrometer (CLAS) shown in Fig. 1. This device is a toroidal multi-gap magnetic spectrometer and is described in detail in the Conceptual Design Report on CEBAF Basic Experimental Equipment [1]. The magnetic field is generated by six iron-free superconducting coils. The particle detection system consists of drift chambers to determine the trajectories of charged particles, Cerenkov detectors for the identification of electrons, scintillation counters for time-of-flight measurements and electromagnetic calorimeters to identify electrons and to detect photons and neutrons. The six segments will be instrumented individually to form six independent spectrometers. Commissioning of the CLAS will begin in the fall of 1996.

The tracking of charged particles is accomplished by three regions of drift chambers that are located at different radial positions from the target. The inner chambers, called "Region I", surround the target in a region of low magnetic field. The "Region II" chambers are somewhat larger and are situated between the toroidal magnet coils in a region of high field, while "Region III" chambers are large devices located radially outward of the magnet. Achieving the design goals for an accuracy of better than 0.5% in momentum and angular resolution of ≈ 1 mrad requires tight constraints on the gas mixture within the chambers. These constraints are discussed in detail in the User Manual and Operation and Safety Procedures (OSP) for the Hall B Drift Chamber Gas System [2] and require control and/or monitoring of the flow rates, mixture, pressures, temperatures and contaminant levels of the gas.

There will be two levels of control in the gas system. The basic flow control loops and alarm system will be hardwired at one level. At a higher level, the system described in this paper will control and monitor the gas system in such a way as to allow:

- Safe operation of the gas system.
- Remote monitoring of signals.
- Remote control of system elements.
- Ease of use via a graphical user interface (GUI).
- Detection of anomalous operational modes, alerting the users of such situations and, where possible, executing automatic procedures to ensure personnel and hardware protection.
- Archiving operational parameters for future restoration and analysis.
- Automation to handle the processing of very large numbers of signals.

The drift chamber gas system control has been identified as one of the highest priority components of the Hall B control system [3, 4].

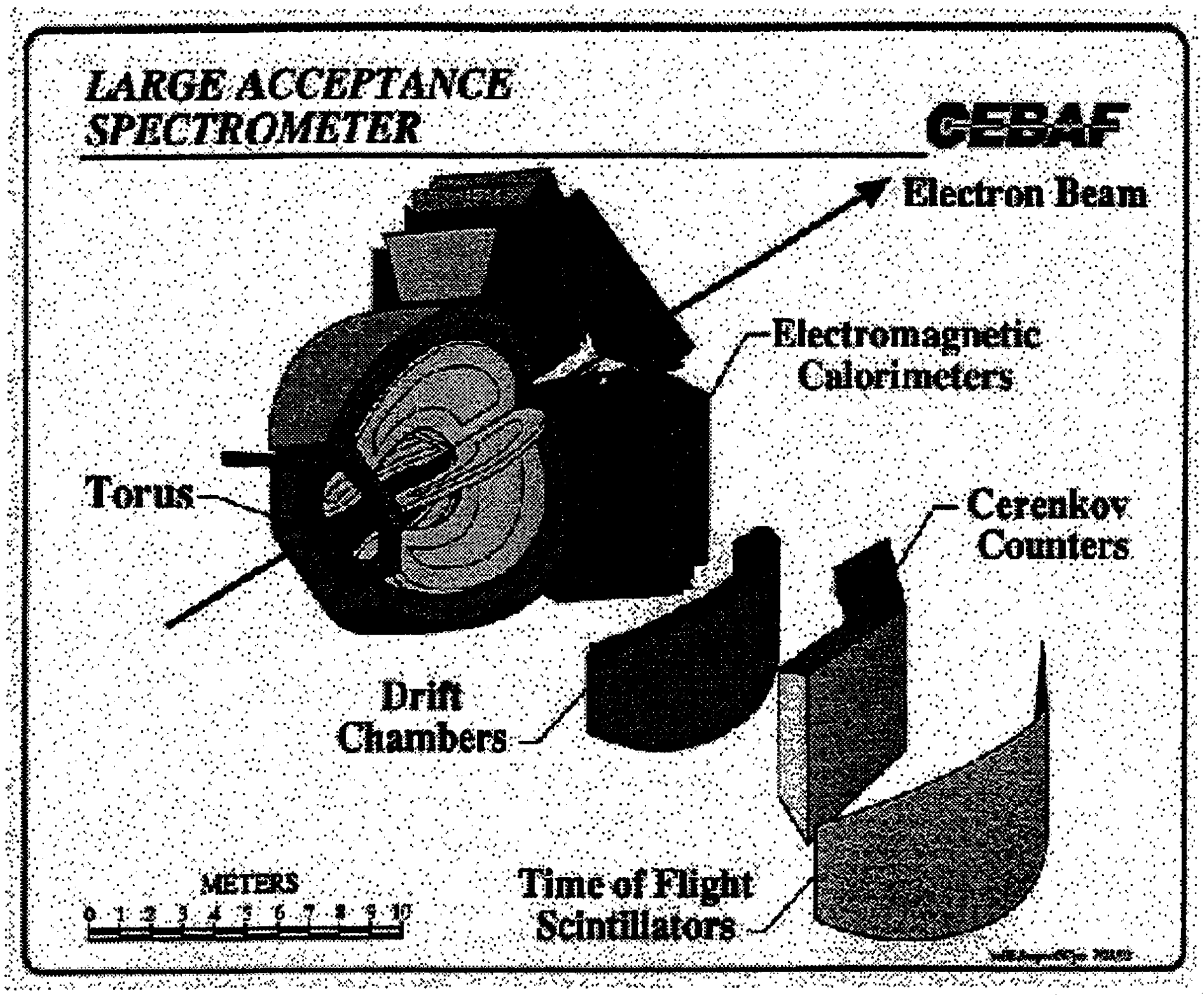

LARGE ACCEPTANCE SPECTROMETER
CEBAF
Electron Beam
Electromagnetic Calorimeters
Torus
Cerenkov Counters
Drift Chambers
METERS
0 1 2 3 4 5 6 7 8 9 10
Time of Flight Scintillators

Figure 1. Exploded view of the CLAS showing the magnet and the various detector components.

THE DRIFT-CHAMBER GAS SYSTEM

The gas system has been designed to meet the stringent requirements outlined in the User Manual and OSP for the Hall B Drift Chamber Gas System [2]. A schematic of the system is shown in Fig. 2. The basic gas system operation consists of mixing, filtering, monitoring, and recirculating (Regions II and III) the gas. When raw gas arrives on site, it is analyzed using a residual gas analyzer (RGA). Once accepted, it is cleared for use with the CLAS drift chambers. During steady state operation, two fresh gases are precisely mixed to achieve the correct relative fraction. The newly mixed gas is stored in large buffer volumes that supply the process loop that contains the CLAS drift chambers. The pressure and percentage of ethane in the gas in the buffer tanks are continuously monitored. As the mixed gas enters the loop, it is first filtered to remove contaminants. An adjustable orifice then drops the pressure and controls the flow. For fine tuning of the gas mixture, additive bubblers are then provided as an option. Directly following the bubblers the oxygen and water content and the temperature and flow rate are monitored before the gas leaves the shed for the hall. As the gas enters the hall, it passes through heat exchangers to damp temperature fluctuations. The gas then enters the chambers where it serves its purpose. At the chamber exit manifolds, the gas passes by safety bubblers. Downstream of the manifolds, the gas temperature and differential pressure with respect to atmosphere are monitored. This pressure reading is transmitted back to the shed where it controls a proportioning solenoid valve in front of a pump. The pump and "throttle valve" pull gas out of the hall while controlling the pressure in the chambers. Once back at high pressure again in the shed, oxygen and water monitors continuously monitor contaminant levels. Finally, a fraction of the gas is exhausted and the rest reenters the cycle above the filters (Regions II and III).

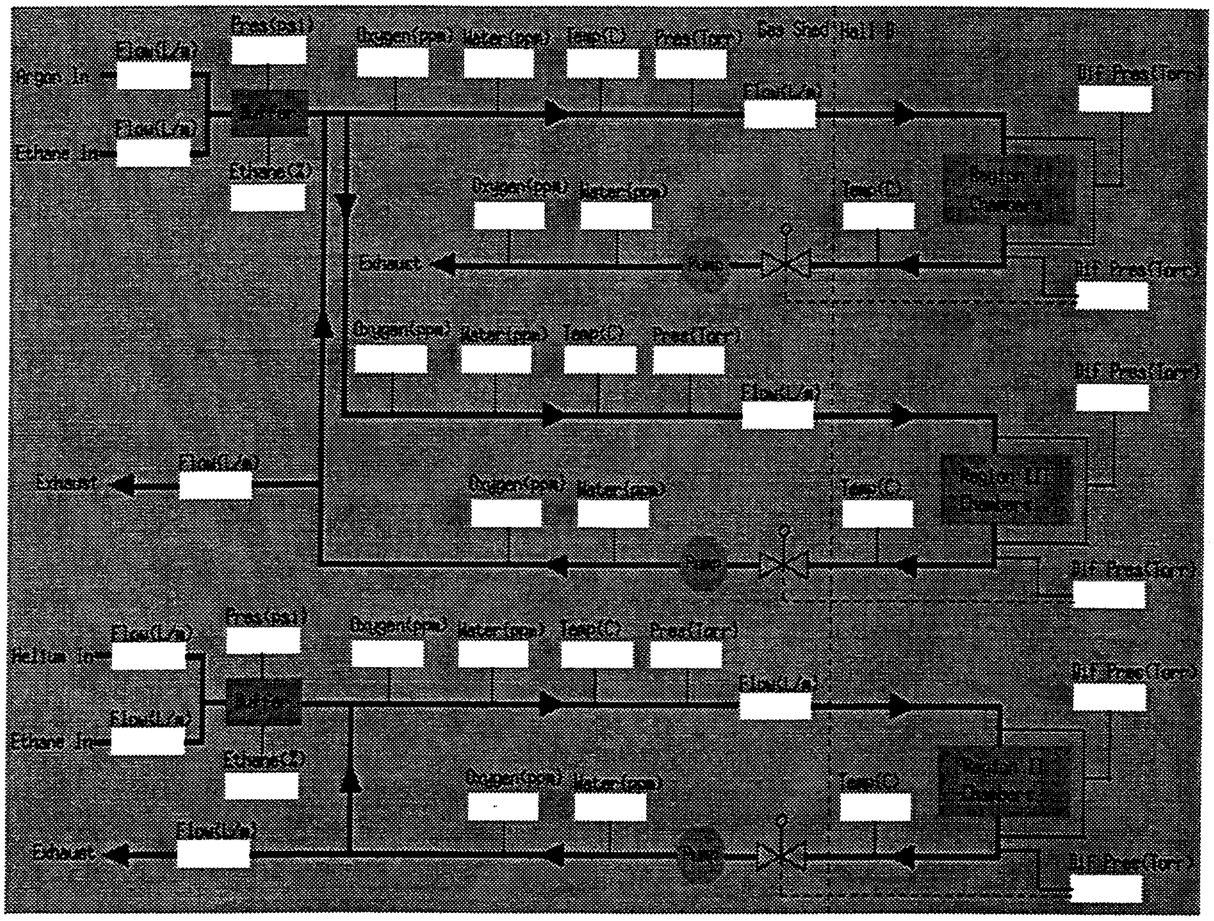


Figure 2. Prototype of the main window of the graphical user interface for the CLAS drift chamber gas system controls. A schematic of the gas system is shown with digital displays to present the analog information.

The various sensors are powered and monitored with three different hardware controllers that have local readout and programming capabilities. The flow controllers and transducers are operated with two MKS 647A mass flow controllers. Pressure sensors are controlled with three MKS 146B pressure gauge controllers. A Panametrics hygrometer is used to monitor temperatures and the concentrations of oxygen, water, and ethane. All of these instruments have RS-232 interfaces for remote control and monitoring. Analog signals are also available for many of the parameters. The instruments have programmable alarms that produce signals that are used to bypass or shut down portions of the gas system. In the event of a failure due to any of a number of conditions, the chamber is bypassed automatically with electronically actuated valves immediately before and after the chambers. These safety features are hardwired and under no software control. The computer control system is needed to monitor parameters and alert the operator of abnormal conditions so that they may be corrected to avoid hardware-activated shut downs. The system is also needed for remote control of the hardware and archiving of parameters for analysis.

THE CONTROL SYSTEM

The control system is being developed within the framework of the Experimental Physics and Industrial Control System (EPICS) [5]. A schematic of the control system hardware is shown in Fig. 3. The operator interface (OPI) is a Hewlett Packard workstation with an X-windows GUI. The input/output controller (IOC) is a Motorola MV 162 single board computer running the VxWorks operating system and mounted in a VME crate. The crate also contains two XYCOM XVME-560 analog input cards and a Green Spring IP-octal 232 industry pack. Communication between the OPI and the IOC is accomplished through ethernet with the EPICS network communication software called Channel Access. The system includes RS-232 interfaces to the two mass flow controllers (MFC), the three pressure controllers (PC), and the hygrometer (HYG).

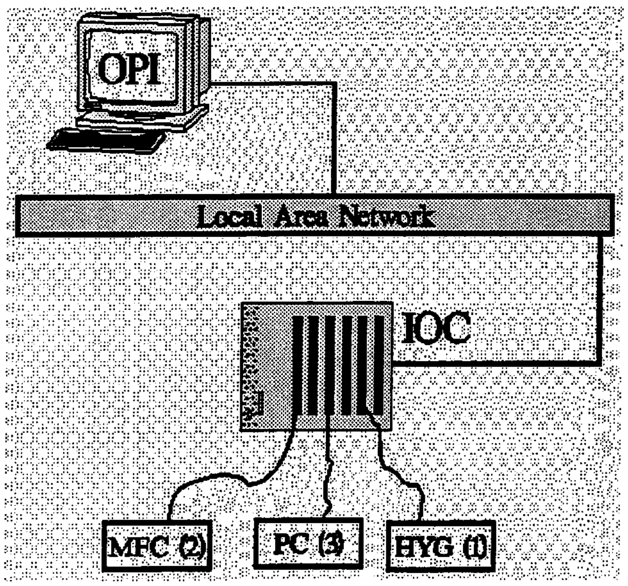


Figure 3. A schematic of the control system hardware showing the operator interface (OPI), the input/output controller (IOC) and the low-level hardware controllers.

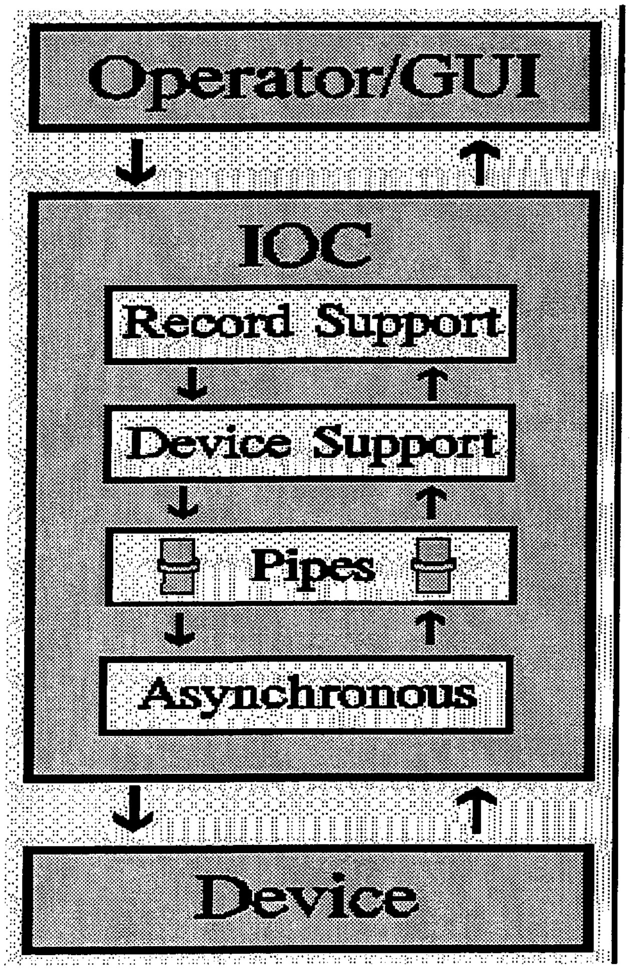


Figure 4: A schematic of the RS232 software interface

Analog Signals
The analog signals from the gas system instrumentation will be continuously monitored with the VME ADCs operating in the 32-channel differential input mode. Device and driver support routines for the XYCOM XVME-560 have been obtained from the SSC and are running on our EPICS development system after some modifications. All of the analog signals are being configured for data logging and many for alarms.

RS-232 Interfaces
Remote control of the hardware and verification of the analog readings will be accomplished through the RS-232 interfaces. The RS-232 device support software has been written and tested for the flow and pressure controllers. The device support for the Panametrics hygrometer will be developed in the near future. An overview of the RS-232 software interface is illustrated in Fig. 4. Standard EPICS analog output records are used to send commands to the device support. Commands without special parameters use a common analog output record, while those with special parameters each have their own analog output record. The device support is split into synchronous and asynchronous parts. This is necessary because EPICS is a synchronous system that cannot wait for a slow asynchronous device to process commands and respond. The synchronous device support is essentially a simulated synchronous device. It utilizes two Tx pipes, which are standard data structures in the VxWorks environment; one to send commands and one to receive responses. These pipes communicate with the asynchronous device support, which is essentially an infinite loop that continuously checks the output pipe for a command. The asychronous device support is spawned as a task in VxWorks, and it is what actually communicates with the device. Once a command is found, it is converted into an ASCII string of a format acceptable to the device. The asynchronous support then waits for a response from the device. Once it is received, the response is placed in the other of the Tx pipes, where it can then be removed by the synchronous device support. Having been removed, the response is placed in a standard EPICS string input record. If any errors are encountered, the device support will instead place an appropriate error string into the record. Once in this record, the response can then be accessed by the GUI.

Graphical User Interface
The GUI is being developed using an EPICS tool called the Motif Edit and Display Manager (MEDM). The main screen of the GUI will consist of a schematic of the gas system with the analog information presented in

digital displays. A prototype of this window is shown in Fig. 1. Along the bottom of the main screen will be a menu to bring up windows for the different RS-232 instruments. These lower level screens include buttons, which are connected to the analog output records and send commands and text displays connected to the string input records for the responses. Examples of these screens are shown in Figs. 5 and 6.

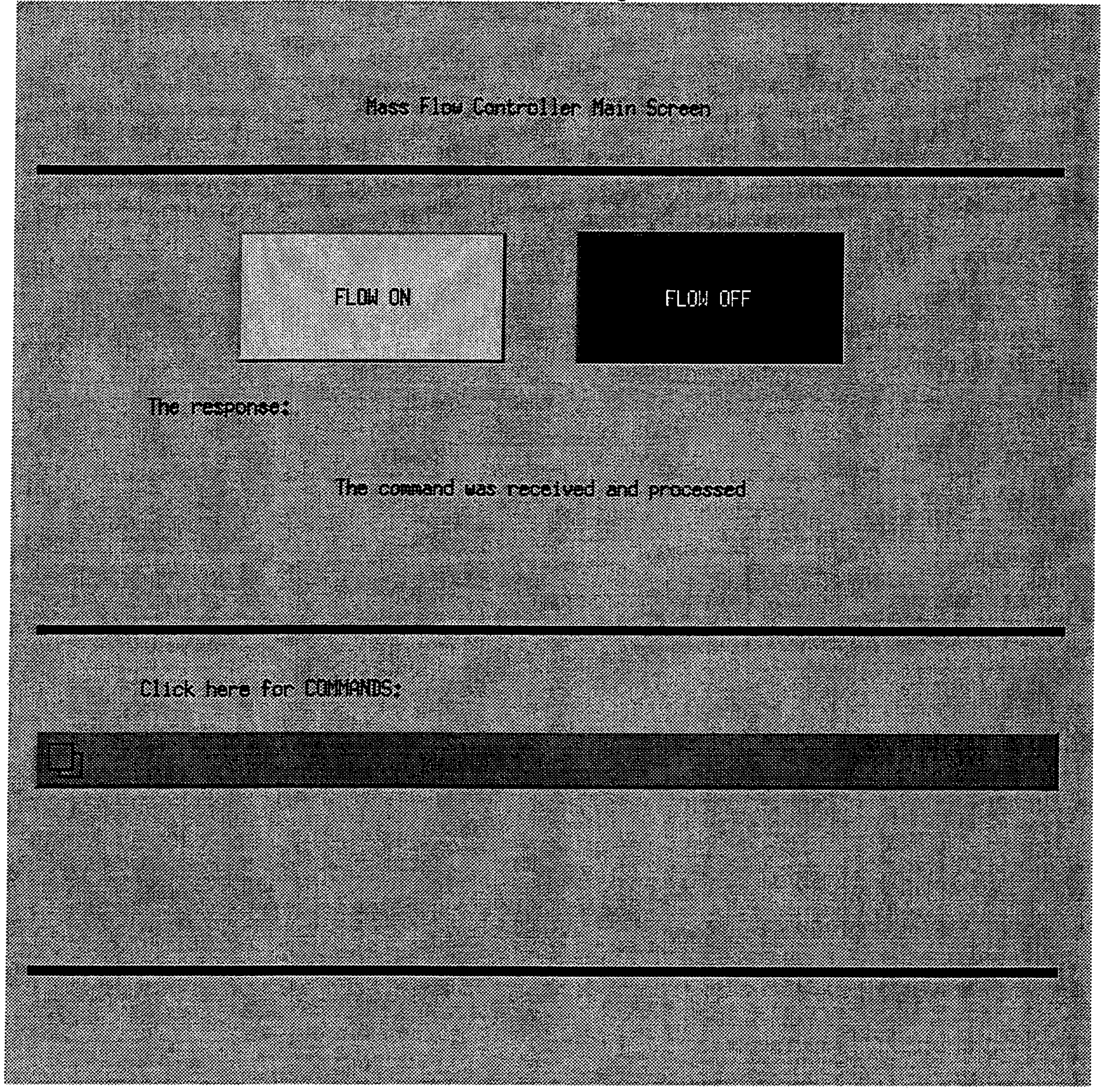

Mass Flow Controller Main Screen
FLOW ON
FLOW OFF
The response:
The command was received and processed
Click here for COMMANDS:

Figure 5: The GUI screen for the mass flow controller.

SUMMARY

A control system is being developed with EPICS for the drift-chamber gas system for the CLAS detector at CEBAF. The gas parameters will be continuously monitored and configured for alarms and data logging. Remote control of system elements will be accomplished through RS-232 interfaces to low-level hardware controllers. Information will be passed efficiently to and from the operator via a GUI. The design of the system is nearly complete. Device support software has been written and tested for all the hardware except the Panametrics hygrometer, and prototype

GUI screens have been developed. Future work includes device support development for the hygrometer, implementation of valve status monitoring, and integration with other components of the Hall B control system.

ACKNOWLEDGEMENTS

This work is supported by the U. S. Department of Energy under Contract No. DE-FG05-88ER40459 and by the University of Richmond.

REFERENCES

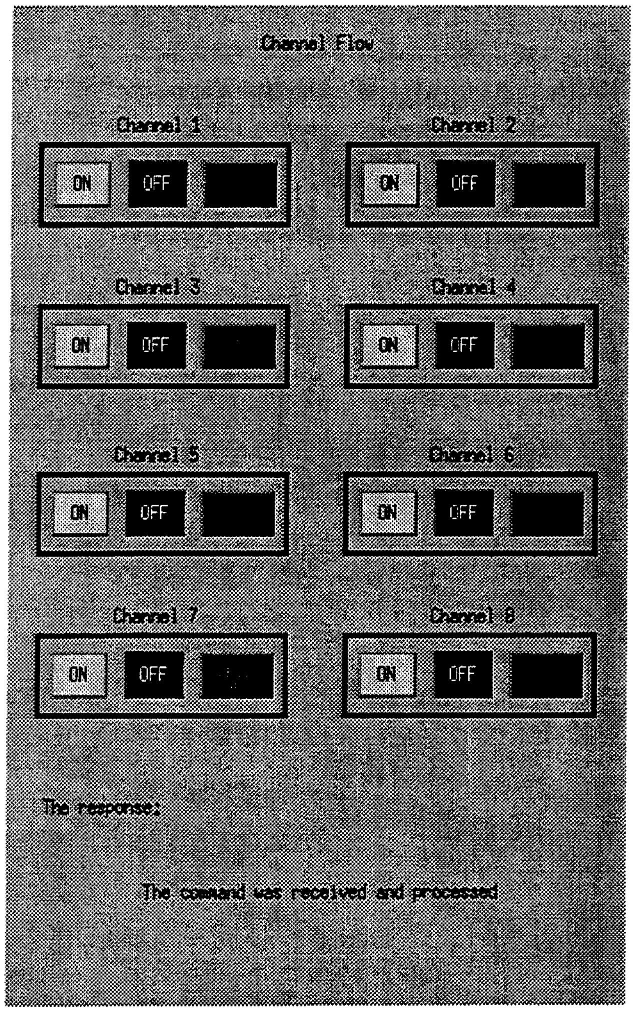
[1] Conceptual Design Report, Basic Experimental Equipment, CEBAF Report, April, 1990.

[2] User Manual and OSP for the Hall B Drift Chamber Gas System, S. Christo,W. Major, M. Mestayer, R. Wines, and Pete Woods, Working Draft CEBAF Report, June, 1995.

[3] Hall B Slow Controls System Requirements Document Overview (SRD), D. Barker and the Hall B Controls Group, Draft CEBAF Report, June, 1995. Information on the system can be found on the World Wide Web at http://www.urich.edu/~vineyard /SRD.book_1.html.

[4] Commissioning the Hall B Experimental Equipment, Draft CEBAF Report, June, 1995.

[5] The Experimental Physics and Industrial Control System (EPICS) is a set of software tools originally developed by Argonne National Laboratory and Los Alamos National Laboratory for the purpose of building distributed control systems. Information on EPICS can be found on the World Wide Web at http://www.aps.anl.gov/asd/controls/epics_home.html.



The command was received and processed

Figure 6: The command subscreen for the mass flow controller.

Software Project for SPring-8 Linac Control

H. Yoshikawa, Y. Itoh, H. Sakaki, A. Kuba, M. Kodera, T. Taniuchi,
K. Tamezane, T. Hori, S. Suzuki, A. Mizuno, K. Yanagida and H. Yokomizo.
JAERI-RIKEN SPring-8 Project Team,
Kamigori-cho, Ako-gun, Hyogo, 678-12 JAPAN

The injector Linac control system for SPring-8 is under development and the first trial will be held soon for testing total performance. We designed this system using object oriented methodology to integrate the operator environment and the database architecture and for easy maintainability. A disadvantage of the object oriented approach is a long lead time for abstraction, modeling and definitions of entities and relations. We represent the status of our software project management and our know-how of rapid application development.

I. Introduction

The control system of the SPring-8 injector Linac has been based on the concept of object oriented design from the beginning of construction. The purpose of this Linac is not only injection to the storage ring but also injection to a new VUV ring SUBARU , the SASE source, a parametric x-ray source and possibly other purposes as they arise.

As a matter of course this software system will be modified continuously as the hardware changes. Thus we must consider not only the functionality of this system, but its maintainability, flexibility and presumed longevity.

II. Applicability of OOP

At the beginning of this software project we discussed which was better - conventional structured or object oriented programming. The former is very mature and many companies have huge resources. If we could write down a complete specification of the software (before the design of hardware devices), some big companies could create the entire software system and it would not be necessary for us to hire programmers. But requirements of our Linac control system are high flexibility and adaptability to future modifications and upgrades without large costs in manpower. That is to say, it would be inevitable that we would have to understand the inner structures of the whole program. The modern belief is that object oriented techniques must be applied as the methodology for management of the complexity of such a large scale program, and thus we chose the OOP paradigm.

One of the disadvantages of OOP is a long lead time. It takes a long time before a schematic of the system for users (operators) appears. We faced the additional problem that our staff was not familiar with the concept. It was predicted that a long time would be required for study of OOP and preparation of a satisfactory class library. However, at the beginning the SPring-8 project schedule was stretched by budget conditions. There was not enough staff and money, but there was time.

The next question asked concerning our use of OOP was whether the scale of the software system was or was not appropriate. When you require the system for control of a small experiment, OOP is too heavy. However, you can still use ready-made class libraries for the GUI and windows.

The number of channels required for control of the Linac are as follows.

The number of signals						
DI	DO	AI	AO	RS232c	GPIB	step motor
1605	749	440	139	3	25	108

The scale is large enough, but the number of kinds of devices is small. For instance, several kinds of magnet power supply are similar to each other. Spatial replication of device structures is the rule, e.g. Q-magnets, ion pumps and beam monitors. The configuration of the Linac is shown in figure 1.

The characteristics fit well with the class hierarchies and inheritance of OOP. Thus we felt that it would be possible to

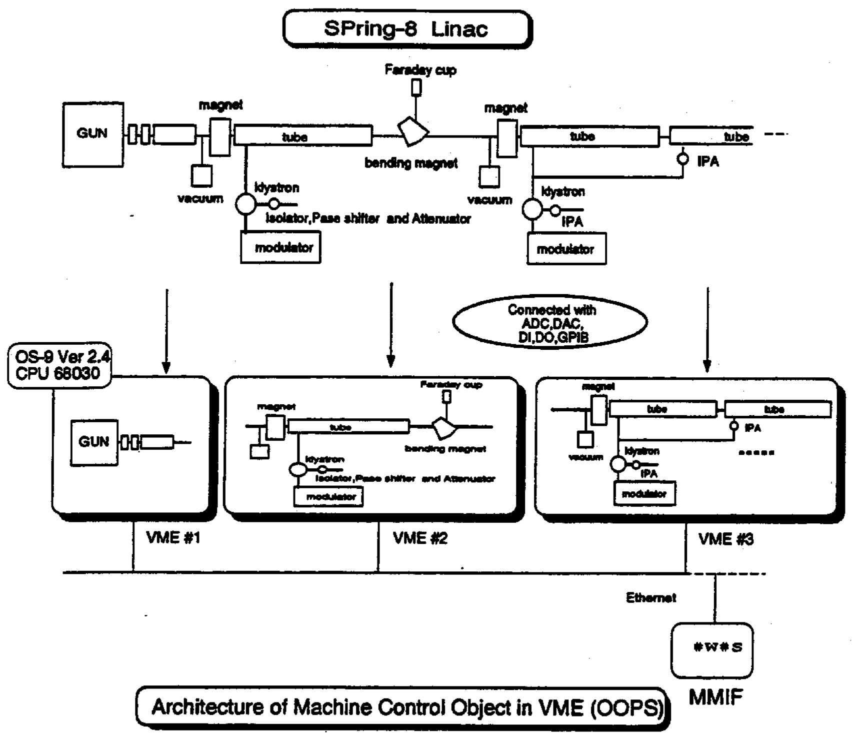


Figure. 1. Configuration of the Linac.

create the required libraries on the appropriate timescale.

III. Progress of the project

Initially, we made an abstraction of the functionality of each atomic device, macro devices and the whole Linac at a software engineering level. The behavior of individual devices is fairly simple. Most of them require only CW operation; and complex feedback (feed-forward for the electron beam) is not needed.

The modulator of the pulsed klystron is an unstable device because it includes discharge elements, but this instability can be hidden from upper level processes, which can consider it as a high voltage power supply.

A. Modeling of the system

We made a common model (MACHINE MODEL) of the status conditions of all devices, as shown in figure 2. We proposed to match the specification of device behavior to this model. The model is event-driven, and a message from some other object is the trigger for a status change. Messages are standardized for all devices inside SCCs (SPring-8 Linac Control Commands). When a device in STANDBY status receives a "GO RUN" SCC, the device acts to change to RUN status. Individual differences in this behavior are encapsulated in the elemental control processes. A macro device, which consists of several atomic devices, operates similarly and forwards SCCs to its internal element devices.

B. Bottom up approach

Code writing began at the level of device drivers. These were written in C due to its potential portability to advanced CPU boards in the future. VME was chosen as the hardware platform as it is popular world-wide and high performance new boards appear daily. Device drivers in C have advantages not only for new CPU boards, but also for new I/O boards, as shown in figure 3.

The process configuration of this system is shown in figure 4. We worked from the lower levels of device drivers, machine control processes and communication processes to the upper level operation process. The behavior of lower level control

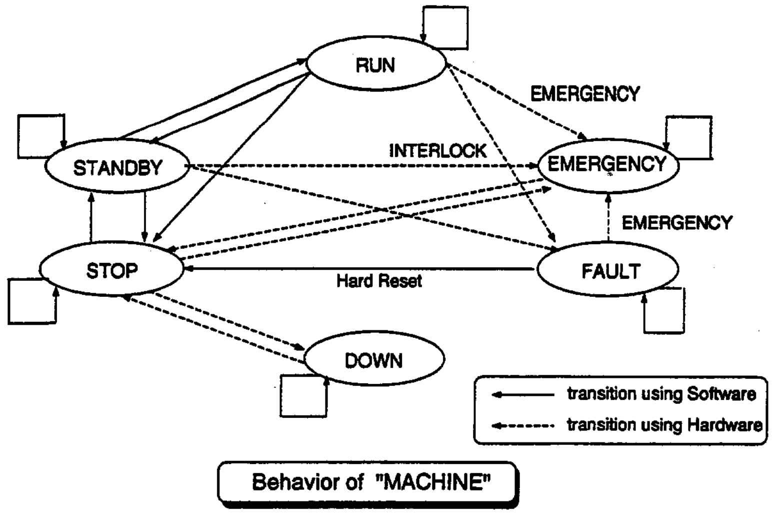


Figure. 2. Status transfer of MACHINE MODEL.

processes depends on hardware design. We required the specification of signal handling to hardware before the software design was fixed.

The communication process is implemented via the original protocol SCD (SPring-8 Control Datagram) [1]. The network attaches directly to the objects, and network transparency is very important. Moreover the communication process is the most important in actual total performance. We isolate the Linac control system segment from other protocols, and only SCC packets are transmitted on it.

The Operation process on EWS (Engineering WorkStations) includes a man-machine interface based on X11 and behavior definitions of device operations. The development environment on EWS is now very sophisticated and many kinds of CASE tools and GUI builders exist. We consider that processes at this level must be modifiable by accelerator specialists and operators to mature.

IV. Framework of control process

The fundamental structure of MACHINE MODEL is shown in figure 5. Receiving of SCCs and interrupts from hardware are recognized by the process as signals. The process checks status, searches the lookup table and jumps to the selected procedure. All input and output except hardware interrupts are dealt with as SCCs. This framework is common to all

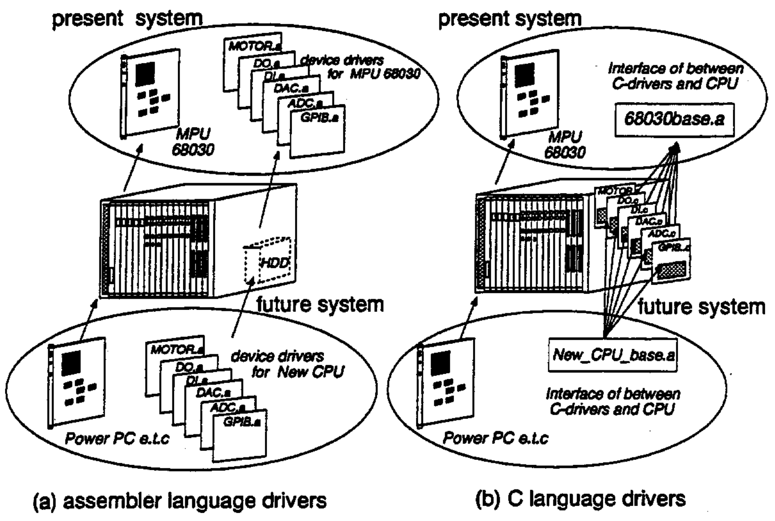


Figure. 3. Differences between assembler drivers and C drivers.

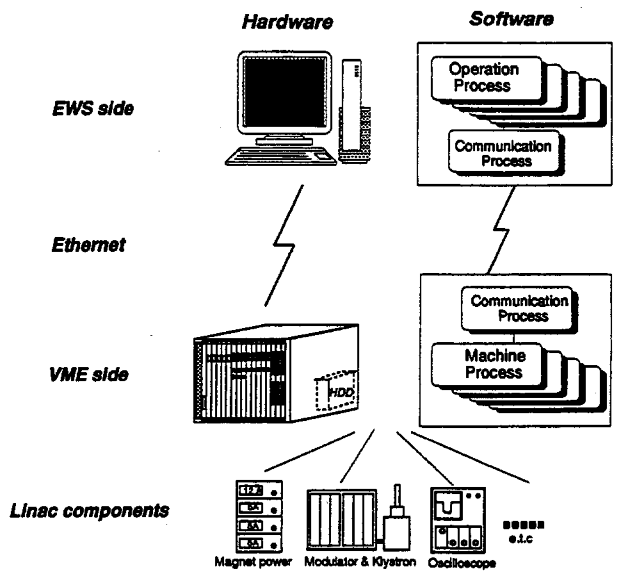


Figure. 4. Hierarchy of processes

processes and is very simple. We represent the framework and the definition of behaviors to programmers, and they write the local procedures. The difficulty in defining behavior is to hide the time sequence of hardware action.

In a most important aspect, we were forced to change the ways of thinking of operators. For example, the modulator of a pulsed klystron has two phases of fault, only High Voltage Down and both High Voltage and Low Voltage Down, in which latter it takes half an hour to warm up the heater of the thyratron and klystron. This is not matched with MACHINE MODEL. We suppressed heat up time in "STOP" status, and the "GO STOP" action includes "heater on". "Warm Up" is defined as a necessary condition of the action to "GO STANDBY" and is independent from identifying of status. Every process on VME and EWS has this same framework. Input to the processes on EWS includes X-window events.

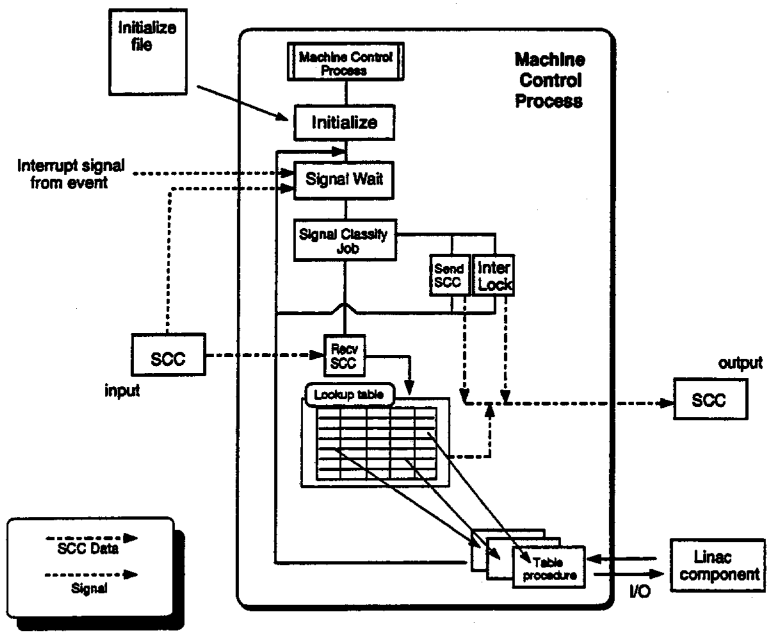


Figure. 5. Framework of control process.

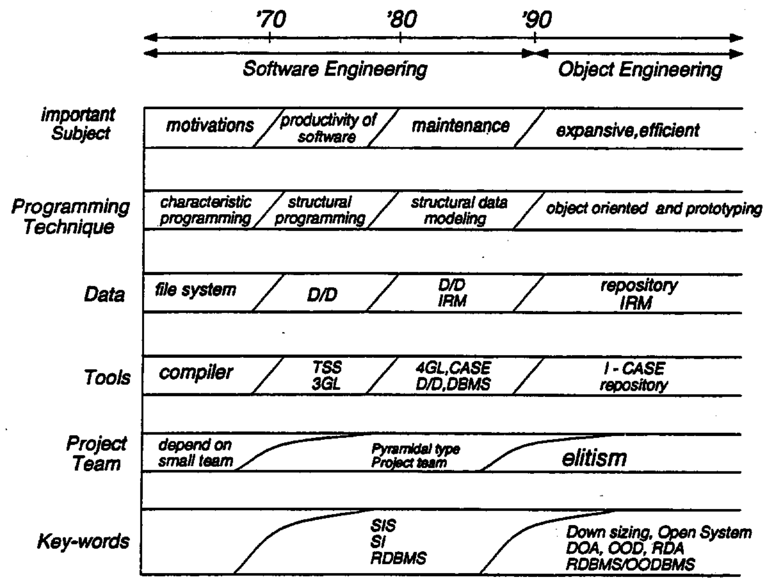


Figure. 6. Trend of software technology.

V. Operation process

Operation processes are located in EWS. This process includes the X-window procedures and operation sequences in the call back functions. We use SoftBench as a CASE tool, and UIM/X as a GUI builder, since it is suitable for combining with SoftBench. A few widgets which we need are written by us, but primarily Motif-based widgets are used.

We prepared three types of man-machine interface program as operator processes. One is a character-based SCC handler for local debugging on VME. It is able to be used through a telnet command from anywhere. The second is a 'primitive' window, which is prepared by the programmer and supports individual device action.

The last is the graphical window, the one to which actual operators interface. On this level, the naked action of MACHINE MODEL is hidden under the X11 environment, and virtual devices appear in front of operators. The virtual devices are objects which have several devices in them, for instance, an energy analyzer (bending magnet, slit and Faraday cup), emittance monitor (Q-magnet, slit and wire grid monitor), or vacuum system (many ion pumps and cold cathode gauges) etc.

The parent process of all operation processes is also one of the graphical operation processes, called LinacMain. LinacMain is the default receiver of SCCs from any VME control process which does not have an operative parent. Finally, this Linac can be operated by only pushing the buttons "STANDBY" and "RUN" in LinacMain.

The control of RF phase is beam-oriented. Previously it had been controlled by a maximum output current condition. However we now detect the phase of the beam wakefield at each accelerating section by a directional coupler near the dummy load, and set the phase of the input power to reverse it. This sequence is written into the operational process "RF system", and when we get fundamental preset values auto-phasing becomes possible.

In the near future many virtual device processes will be written to make the environment of operators more comfortable. Moreover, the modifications for injection into SUBARU, and for beam transport to SASE will be done in the next few years. In this situation the development environment of the software is very important. Use of CASE tools is inevitable, and the advancement of operating system version will be discussed at the level of revision management and efficiency of resource use.

VI. Strategy for construction

The number of persons in the Linac group is eleven, of whom six are working on software. The software staff consists of an overall coordinator, VME programmers, communication process designers, database builders and a handler of visualization tools. By September 1996 all programmers will have become GUI writers.

The first phase of the conceptual design was discussed by only a few members, whose initiative led to understanding of the OOP concept and design policy for others. Each member designs his part, and the team has a discussion to review it. In this phase, it is important to decide not only the functionality but also the structure of the program. Finally we present these prototype codes to programmers, and we require them not to change the structure, except for local functions of individual procedures.

We do not have enough manpower to write all the code of this system, but the job sharing which results can be highly cost-effective; it also helps us to keep a consistent design policy from beginning to end. It took two years to establish the prototype code of MACHINE MODEL. The fundamental structure of MACHINE MODEL has required little change, though several options for adaptation to the hardware design of devices have been added. As a result of OOP such modification is very easy after the plan of modification is chosen.

VII. Conclusion

Our project has had a software engineering history as shown in figure 6. We had to learn the concept of the new methodology, and had to change our way of structured programming. For a software project such as this, to keep a consistent policy and sense of design is very important for purposes of continuity. If you do not have usable class libraries, the project manager must endure a long lead time. But in cases where the scale of the system fits OOP, the final total load is smaller than in conventional structured programming (figure 7). If you can find good class libraries, which your system is able to use, the necessary manpower is drastically reduced.

References

[1] H. Yoshikawa, Y. Itoh, K. Tamezane, Y. Sakaki, M. Kodera, H. Yokomizo. *"Class structure of the Injector Linac control system of SPring-8"* Nucl Inst & Method in Phys. A352, 216-217 (1994)

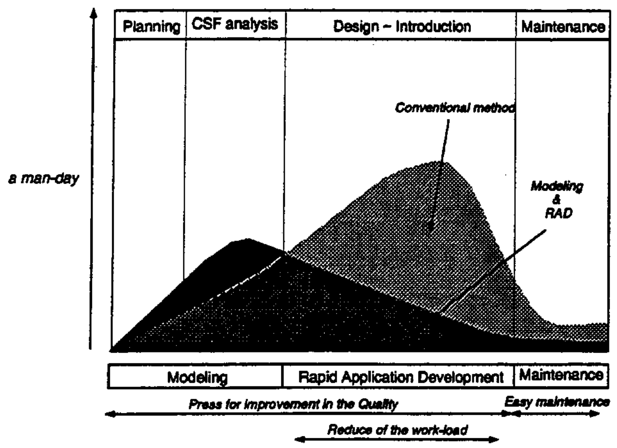


Figure. 7. Manpower at each phase of a project.

Class Analysis and the Object Model in the KEK PF Linac console

Isamu Abe and Kazuo Nakahara

KEK, National laboratory for High Energy Physics
Oho1-1, Tsukuba-shi, Ibaraki 305, Japan

Abstract

The Linac device controllers (front-end computers) and mini-computers for the KEK PF, which has been operating for more than ten years, had to be renewed during 1993 because there was no longer support from the hardware manufacturers. In 1994, the B-Factory project at KEK was approved and will start from 1998, and this requires the Linac to be upgraded from 2.5GeV to 6GeV. The control system also needs an increase in size for the additional klystrons, magnets, vacuum systems and so on At the same time, the console programs must be modified to be suitable for B-factory operation. In the modification phase, in order to reduce the maintenance and development costs, some of the device controller SBCs (Single-Board-Computers) will be replaced by PLCs (programmable logic controllers). Investigations[1] were launched late in 1994 in order to determine the I/O stability and network possibilities. In this paper, the software for the device layer controller (PLC) and the human interface layer to the B-factory are discussed on the basis of the Object Oriented Concept.

1. Introduction

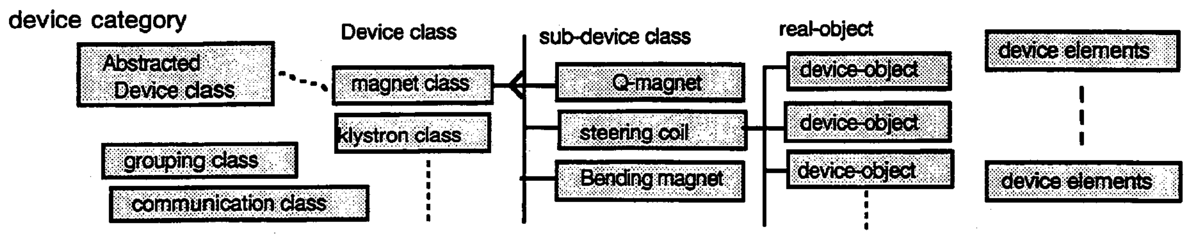
An object-oriented analysis and design[2] of a human interface (operators console system) have been adopted for the PF Linac. The old console system has a graphic display which was written using FBHG (Fujitsu Basic High Grade) on DOS/Windows (Japanese version). It has not been very simple to change the display and its programs, which were written in structured FBHG language. To change the conventional language to a new OOP one requires a big paradigm shift and there are two aspects which are both good and not so good. One is higher productivity with the OOP, although the learning cost for users is also higher. Some disadvantages of the OOP are also becoming clear. We are therefore providing a root, or super class, from which one can derive objects for users (developers / operators) without any programming on the new GUI (graphical user interface) windows. Users can make control and display windows very easily and quickly by just copying a visual object from its mother class, while inheriting methods and properties. After naming the object and changing some properties if necessary, it can be run as a flexible user program on the Linac console.

2. Classes in an accelerator domain

For an entire accelerator domain, a rough object-oriented analysis was made,and then the control-system design proceeded step-by-step based on an OOA (analysis). The OOA and OOD (design) were repeated several times for each field in the accelerator domain. As a result, we divided the accelerator control system into to two major categories: 1) device category and 2) generic category. The generic category can be divided into classes: the data and information process class, the generic tasks class, the beam physics class, and the GUI class [3]. In this paper the device class and GUI class are mainly discussed.

3. Device class

The Linac is composed of many devices, such as magnets (beam transport), klystrons (RF system), injection system, beam monitors (Beam Position Monitor, Core Monitor, Profile Monitor, etc.), vacuum system, trigger system, control system and safety system, which are located along its length. These grouped systems, which have physical-device objects, belong to the device category. The device category has several root classes and/or super classes. Generally, it has subsystems as a child class. In most cases, the super class has a definite behavior, properties and messages to be handled independently. When there are many similar device objects, they can sometimes be classified in the same group in a bottom-up approach. Each object is derived from its mother class. The magnet and the power supply, which are geographically located along the beam line, are typical examples. Once they are well defined, these classes and objects do not have to be changed very often; it is only necessary to install or remove objects when the Linac is physically modified. We may find a common or standard class which might be shareable in several laboratories if it is well analyzed. In 1989, the device and access procedures in an accelerator domain were well analyzed and defined at CERN as reported [4] at ICALEPCS'91. The CERN device protocols were installed in the device-layer computers. In our system, we will attempt to analyze objects in the near future, so as to distribute them on the two layers of the computer system through a network, using the emergent CORBA, OLE or something similar. The root or super class, sub-class, and device element relations are shown below:



4. Generic category

4.1) Data and information class

The data and information (with commands) exchanged between the devices or equipment of the accelerator are defined as objects separate from device objects. Data which is sent to or received from device classes are supposed to be processed in some way at the operator console or by a data-base system or generic task class. Normally, data must be processed in a way that is already known, or decided by an expert based on experience or knowledge. As an example, magnet and vacuum data are commonly displayed as real-time data or a history graph at the operator's console, and then sent to an upper/lower level check filter for the alarm system. Since it is already known how to process most of the accelerator data, we can make data-process classes rather easily abstracted as re-useable ones so that data can be collected as encapsulated objects in a data base.

4.2) Generic task class

There are no actual devices in the generic task class; it is a type of heuristic knowledge class: how to operate an accelerator, or how to run each device. There are many ways to run the Linac and each person can have his own methods to handle the devices or the accelerator to obtain beam. Although it is difficult to find a standard or common class in this generic task, a typical procedure must be defined for the actual operation of the Linac. There are very few derived sub-classes in this category. Each accelerator must have its own class for every operation or diagnosis; there will then be a few shareable common classes among the different laboratories concerning generic tasks.

4.3) Beam physics class

Mathematical models are commonly used for beam dynamics calculations and their handling on the networked computers. Some people use OOP methods for modeling, while others do not. Since these are carried out at the accelerator-design phase, they are not considered here.

4.4) GUI class

In the GUI class, the object-oriented approach has been a most effective way to make an easy-to-use and flexible graphical user interface on Windows. The accelerator GUI requires some graphical icons, which are for abstract devices on the Windows display. The graphical icon has derived properties and methods based on its mother class. Two layers of the mother and child classes represent a more convenient way in actual operations or in the development phase of the GUI. A deeper class should be related to the mother class, behind which it does not appear on the GUI. Simple changes in the properties and methods are easy to make in the property lists or windows. There are necessary GUI elements, which should be abstracted objects or those encapsulated for the Linac operator:

1) Device_Icon, Operational_sw/lamp_Icon,
2) History_graph, Two-dimensional_graph, Charts
3) Help_menu(www viewer), Text_display,
4) Video_display,

These Icon or GUI elements must be a re-useable mother class which can inherit many useful properties. Some commercial software packages exist which could match our needs.

5. Device layer controller

The magnet and vacuum systems used to be controlled by SBC (single board computer) local controllers which were connected to CAMAC. CAMAC had been running until it was replaced by the VME system. Since we have tried to reduce maintenance costs, the replacement of the old SBC with a PLC was investigated. Although the PLC was originally not intended for a high-speed multi-purpose machine interface, being used in industry as well as other fields, the DAC and ADC modules are sufficiently stable for use with power supplies. The performance of the PLC is almost good enough to control not only the magnet power supplies, but also the vacuum systems and klystron modulators as a low-speed control. The cost is less than that of other interfaces, such as VME or CAMAC, and maintenance is easier. Since most of the PLCs can be connected to Ethernet, it is now possible for them to be controlled from host computers, even though each maker has its own network. The PLC which we have selected has two CPUs: one is for I/O and the other is for communications. TCP/IP and UDP are available protocols which are used to communicate with other computers on Ethernet. The PLC has its own internal programming and tools, and acts as an intelligent controller using ladder logic. Even from Windows 3.1 with a Pentium 90MHz CPU, a 10 to 20 per second refresh rate is possible with the PLC. That is adequate scan speed for an operator when he adjusts the magnet current or other variable.

6. PLC class

PLCs are produced by several companies and there are no standards among the manufacturers. A PLC can support various types of I/O modules: such as DI/O, DAC, ADC and IRQ. The PLC can be a member of the super class at a device-layer computer and the I/O modules are defined as belonging to the sub-class. The PLC super class must communicate with the PLC communication class or friend classes. In the PF Linac, the abstracted device classes are also set on the human-interface layer so the PLC communication class could communicate with the PLC device objects which are distributed at the device layer through the network.

7. Implementation

The PLC used for the Linac device controller will be connected to about 400 sets of magnets, 100 sets of vacuum systems and 60 sets of klystrons, which are slow devices to control. In late 1995, a prototype will run the vacuum system at the 2nd sector of the PF Linac. In advance of the hardware system, the first version of the software development has been completed, based on object models using the OMT (Object Making Technique). Device classes (klystron, magnet, vacuum, gun, monitor and others) are defined and coded as a super class on the control pack. The instances (real objects) are derived from their control packs on the GUI windows: for example, the magnet class has its sub class (such as Q-magnet, STC and bending-magnet). Each device class may use a PLC object for intercommunication. It is also related to the other super (or friend) class. Since Visual BASIC has a simple inheritance system, a common mother class has been coded based on the OMT using Visual C++ and was made as *.VBX tools on Visual BASIC. These classes were developed as a control pack which can appear in the toolbox in Visual Basic when it is in interpreter mode. Visual BASIC has limitations in memory size and execution time when many objects are created in the Windows derived from the mother class. A more objective language having better OOP features is now being tested in order to solve these problems.

8. Results and conclusion

Adequate speeds for the control devices were achieved by object oriented programming in the GUI and at the device layer. We could actually make common classes for the PLC device objects and the abstracted GUI objects. The standard class has resulted in a flexible control system at the console of the PF Linac. The OOP also provides users an easy way to make modifications and to produce a highly productive control system for accelerator users or machine operators.

References

[1] Feasibility study for PLC as a device controller, A. Shirakawa, Linac conference 1995, Osaka
[2] Object-Oriented Modeling and Design James Rumbaugh, Prentice Hall, Inc.
[3] GUI object model in Accelerator control
I.abe, K.Nakahara, M.Mutoh, Y.Shibasaki, Linac conference 1995, Osaka
[4] Control Protocol: The proposed new CERN standard access procedure to accelerator equipment.
G. Baribaud and etc. ICALPCS91, p591

Object Oriented API Layers Improve the Modularity of Applications Controlling Accelerator Physics

T. Birke, R. Lange, R. Müller

Berliner Elektronenspeicherring-Gesellschaft für Synchrotronstrahlung m.b.H.
(BESSY), Lentzeallee 100, D–14195 Berlin, FRG*

Abstract

With EPICS the architecture of core software and standard control programs is largely defined by the given tools. The installation of EPICS at BESSY II focuses attention on configuration and improvement issues. This does not cover the applications measuring and controlling accelerator physics. Despite similar problems at different sites these types of programs are seldom portable due to the low degree of modularity or encapsulation. At BESSY dedicated API layers written in C++ have proven to be a useful means to hide implementation details: An event-driven interface to a graphics server allows one to write programs with a graphical user interface on a windowing system (now: X11/Motif) without detailed knowledge of that system. Device objects on top of the equipment access layer (now: CA/*cdev*) implement hardware specific control methods. This facilitates device independent coding of measurement sequences.

I. Introduction

The 'Berliner Elektronenspeicherring-Gesellschaft für Synchrotronstrahlung m.b.H.' operates a 0.8 GeV storage ring dedicated to the generation of synchrotron light in the VUV and soft X-ray region, BESSY I.[1] The successor BESSY II has been designed as a high brillance synchrotron light source in the VUV/XUV region with 1.7 GeV nominal beam energy. Sixteen DBA cells provide alternating dispersion-free straight sections of high and low horizontal beta function. This machine is presently under construction at Berlin-Adlershof and scheduled to start routine operation in 1998.[2]

The replacement of the original control system of BESSY I [3], [4] was completed in 1993. [5] The architecture of the new system embodies the 'standard model' of distributed control system design. Due to the restrictions imposed by the replacement process it contains some peculiarities. As a novel feature full advantage of embedded controllers and the properties of the CAN fieldbus is taken.[6], [7]

Apart from an attempt to save investments and know-how where appropriate there are no intrinsic boundary conditions on the control system design for the new machine BESSY II. We decided not to proceed with our own development but to base the system core on EPICS.[8] As part of the overall architecture the extensive use of the concept 'embedded controllers as CAN field bus nodes' remains.[9]

II. Context of Application Development

The control system applications needed for the operation of BESSY II are coupled to the underlying control system core by the 'application programming interface' (API) and the data model involved. That is sufficiently universal to be able to profit from the experience and investments made during the project BESSY I. A new restriction is that developments now have to suit the environment of EPICS tools.

A. *Experiences and Investments at BESSY I*

Concepts or solutions that have proven their usefulness are planned to be taken over at least partly.

• The data model at BESSY I is very simple. It requires polling, only synchronous 'get' and 'put' operations are implemented. Therefore the callback mechanisms of 'channel access' (CA) provides a big increase in functionality. On the other hand the API for the BESSY I equipment access is already based on exchanging messages with devices. This abstraction level of 'devices' and their data structures is not provided by CA.
• The 'graphical user interface' (GUI) has been implemented by a graphical server encapsulating complete representational aspects. It provides synoptic views, control panels and window hierarchies. Clients communicate with the graphical server on an event driven API exclusively about requests concerning application variables.[4] The most versatile client is an

*eMail: birke@bessy.de, lange@bessy.de, mueller@bessy.de

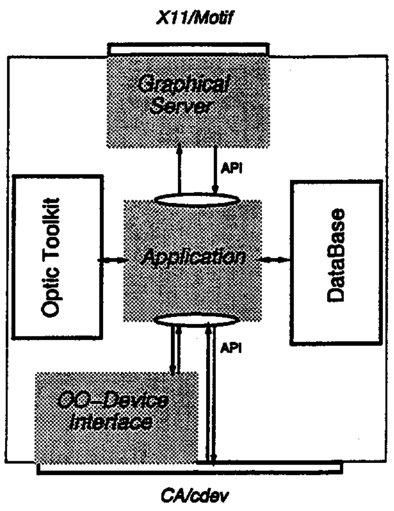


Figure. 1. Application Program Support

interpreter that allows for any equipment access call. Simple tasks can be easily implemented in a scripting language embedded in the representation description file of the graphical server–for the end user somewhat comparable to tcl/Tk scripts.

- A multi purpose hardware driving utility mainly for backup/restore and magnet conditioning has been developed. It implements device classes according to their data sets and meaningful driving methods. Handling of lists of devices, lists of lists and machine data files (snapshots, references) are supported. For the needs of BESSY I macro programming of a single application program that parses command input [10] has been sufficient (Fig. 3).

B. Framework of EPICS

- The *cdev* API [11] provides access to devices by using asynchronous operations and monitor callbacks (features present in CA). At the same time *cdev* hides implementation details of the underlying package(s). *cdev* has been chosen by several major laboratories (EPCS SOSH Workshop, CERN, 5/95) as the candidate for a common calling convention that promises to facilitate sharing of accelerator control applications. Wherever possible applications will be based on the *cdev* API.
- With the EPICS extensions (*medm*, *km*, *dp*, *alh...*) the standard requirements for accelerator operation are covered by sophisticated and high performance tools. However since they are solutions to specific problems on their own the construction of a homogeneous and concise GUI is a challenge. With *medm* it is easy to build fancy screens but there is no support for global variables that affect process variables (e.g. sector number/corresponding corrector power supplies). Support for the construction of hierarchies is minimal (only by the related display button). Resolution of device names into functional parts and the assignment of control panels corresponding to the appropriate device class is impossible. There is no major problem in building synoptic displays with tcl/Tk scripts and in launching the different EPICS tools or *medm* screens graphically in the appropriate context. The main deficiency of this solution is the rapid decrease of maintainability as the GUI evolves.

III. Conceptual Issues

Even if subsets of application programs fulfill the requirements for modularity there are important areas that are usually hard to maintain (Fig. 1):

- Consistency of relatively static configuration data is quite easily provided by making it mandatory for the programmers to use the database (possibly simply stored in central filesets). For BESSY II the reference data will be supplied by the Oracle DBMS.
- Consistent results of model calculations or frequently used standard algorithms can be achieved by providing a centrally maintained optics toolkit and high level software library. Optics calculations will be performed in the context of Goemon [12], a C++ class library. Other utilities are to be developed as they are needed.

- A GUI based on the Motif toolkit requires a considerable amount of understanding. The success of tcl/Tk shows that solutions that do not presume a knowledge of Motif are a reasonable goal. Furthermore even with the support of a

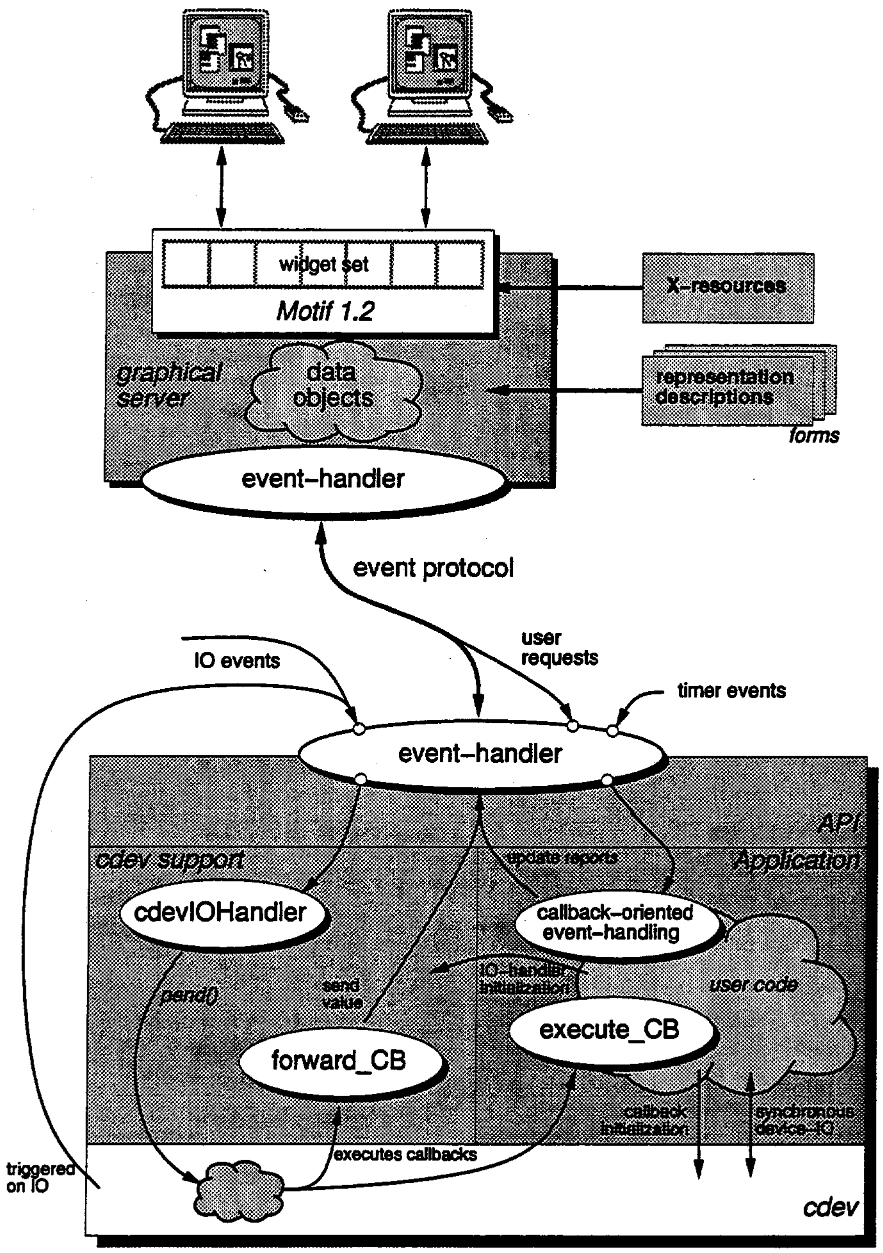


Figure. 2. Graphic Client Event Flow

commercial GUI builder it takes a considerable effort to keep all aspects of the graphical representation localized. The solution of designing an API free from presentational elements and providing network access to the graphic server of an X–display is presented in this paper.

• Hardware control and conditioning programs usually have to deal with all the hardware peculiarities and exceptions. It is described below how the treatment of these hardware specifics is encapsulated in a device object library.

IV. Graphical User Interfaces

The graphical server has been described elsewhere [4] in detail. The structure of this program can be summarized by three blocks: graphical entities based on some windowing toolkit, dynamically created data objects that link presentational and application variables and an event driven interface to the client programs (Fig. 2).

As a result of developments carried out by an external company the presentational layer of the graphical server is now constructed with Motif widgets (formerly InterViews). This improves dramatically the potentialities of this tool: external

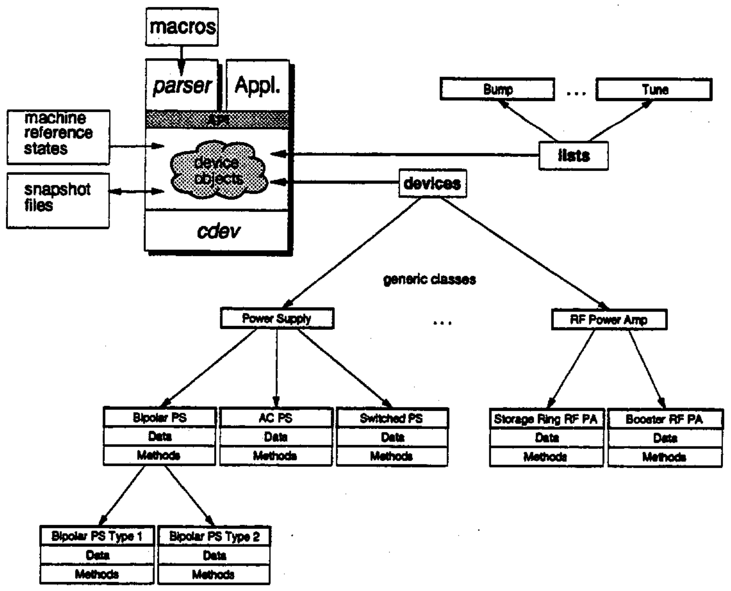


Figure. 3. Sketch of Device Object Environment

widgets are easily introduced and control of specific behaviour is possible with the mechanisms of Motif resource handling. To this is now added the possibility to switch between different presentational entities according to the content in the actual representation description file (form). The forms are requested and evaluated by the graphical server at run time – a feature already present in the previous version of the server .[4] Clients of the graphic server are classically event driven and relatively simply structured. The API calls allow the building of a connection to the server that initiates transfer of application variable data structures and mapping of the window(s). Code is executed according to the incoming events, results are reported by generating events. All client events are grouped into the basic types:

- elementary connection handling (sanity of communication objects)
- user requests (interaction with the GUI)
- update reports (results, changes)
- *cdev* callbacks (frequently simply converted to update reports)

With this API application programmers can take advantage of a powerful GUI with literally no idea about Motif, as long as the widget resources are centrally maintained. This is in contrast to (professional) Motif GUI builders or other toolkits where at least the Motif architecture has to be known to the end user. The generic client of the graphical server named *action language interpreter* [4] is presently adapted to the new EPICS environment by implementing the full *cdev* functionality. It is planned to merge the advantages of GUI elements generated by the graphic server and the generic client programmed with forms, with the fancy screens easily produced and modified by *medm* to an operator interface of high functionality. The former will provide synoptic views and flexible window hierarchies, embed the different *medm* screens and launch the other tools (*km*, *dp*, ...) with the proper parameters.

V. Device Objects

Originally this tool was developed from the need to write an 'intelligent' store-and- load utility for snapshot files that performs conditioning, state modifications of devices etc. According to experience running programs based on the plain equipment access API, with otherwise nicely structured code, are frequently ruined by the exception handling of specific devices. Device objects have been created that adress these problems. They consist of device data sets (status, set point, read back...) and control methods. These methods perform not only the downloading of values or commands, but also complex operations: checks of task success, attempts to put the device into a proper state, possibly the untertaking of repair or surveillance operations. More complex device objects are constructed from simpler objects through inheritance. Collections of these device objects form nested lists. We plan to implement specific lists (e.g. 'bump') inherited from the base list. Devices in these lists are not simply controlled sequentially but 'simultaneously' in 'small steps'. Action is only performed when all the devices in the list are in a proper state. Today we find similar data structures elsewhere in the EPICS environment: the data sets and lists correspond to the *cdev* device attributes and compound devices. The principal application 'backup/restore' could be implemented by BURT tasks embedded in shell scripts. The device-specific driving methods still offer additional functionality to the application programmers that cannot be implemented in a site- independent data handling layer like *cdev*. With the API to the library of device objects we plan to ease the coding of measurement and correction programs and enhance maintainability at the same time.

VI. Summary

The rapid installation of the EPICS tools provides immediately basic functionalities for remote control and surveillance of the connected pieces of equipment. Saving of manpower is invested into the development of tools that improve modularity. The graphic server provides a reasonable balance between coding simplicity, representational flexibility, performance power and demands on system resources. It shields the application programmers from the painful task of learning Motif. Specific hardware control methods and device failure condition handling is encapsulated within the library of device objects . The simple and uniform API will enable the programmer to write conditioning or measurement programs without knowing the hardware specifics.

References

[1] S. Bernstorff et. al., Physica Scripta, 36, 15 (1987)

[2] E. Jaeschke for the BESSY II design team, Conference Record of the 1995 IEEE Particle Accelerator Conference, Dallas, to be published

[3] G. v. Egan-Krieger, R. Müller, Proceedings of the 2nd European Particle Accelerator Conference, Nice, pp. 872–874, 875–877 (1990)

[4] R. Müller, H.–D. Doll, I. J. Donasch, H. Marxen, H. Pause, Conference Record of the 1991 IEEE Particle Accelerator Conference, San Francisco, pp. 1311–1313 (1991)

R. Müller, Proceedings of the 3rd European Particle Accelerator Conference, Berlin, pp. 1167–1169 (1992)

[5] G. v. Egan-Krieger, R. Müller, J. Rahn, Conference Record of the 1993 IEEE Particle Accelerator Conference, Washington, pp. 1887–1889 (1993)

[6] Road Vehicles – Interchange of Digital Information - Controller Area Network (CAN) for High Speed Communication, Document Iso/DIS 11898, International Standardization Organization (1991)

[7] G. v. Egan-Krieger, T. Stein, J. Rahn, Nuclear Instruments & Methods in Physics Research A 352, pp. 204–206 (1994)

[8] L. Dalesio, J. Hill, M. Kraimer, D. Murray, S. Hunt, M. Clausen, C. Watson, J. Dalesio, Nuclear Instruments & Methods in Physics Research A 352, pp. 179–184 (1994)

[9] J. Bergl, B. Kuner, R. Lange, I. Müller, R. Müller, G. Pfeiffer, J. Rahn, H. Rüdiger, these Proceedings (ICALEPCS , Chicago, 1995)

[10] R. Lange, Diploma Thesis, FB 20, TU-Berlin, 1993

[11] J. Chen, W. Akers, G. Heyes, D. Wu, C. Watson, these Proceedings (ICALEPCS , Chicago, 1995)

[12] H. Nishimura, Nuclear Instruments & Methods in Physics Research A 352, pp. 379-382 (1994)

The Self-Describing Data Sets File Protocol and Toolkit[1]

M. Borland, L. Emery
Argonne National Laboratory; 9700 S. Cass Avenue; Argonne, Il 60439

Abstract

The Self-Describing Data Sets (SDDS) file protocol continues to be used extensively in commissioning the Advanced Photon Source (APS) accelerator complex. The SDDS protocol has proved useful primarily due to the existence of the SDDS Toolkit, a growng set of about 60 generic commandline programs that read and/or write SDDS files. The SDDS Toolkit is also used extensively for simulation postprocessing, giving physicists a single environment for experiment and simulation. With the Toolkit, new SDDS data is displayed and subjected to complex processing without developing new programs. Data from EPICS, laboratory instruments, simulation and other sources are easily integrated. Because the SDDS tools are commandline-based, data processing scripts are readily written using the user's preferred shell language. Since users work within a UNIX shell rather than an application-specific shell or GUI, they may add SDDS-compliant programs and scripts to their personal toolkits without restriction or complication. The SDDS Toolkit has been run under UNIX on SUN OS4, HP-UX, and LINUX. The application of SDDS to accelerator operation is being pursued using Tcl/Tk to provide a GUI.

1 Self-Describing Data

As used in this paper, the concept of self-describing data starts from the recognition that when a scientist thinks of data, he typically associates a number of attributes with it. Among these are

1. The name by which the data is known.
2. The units of the data, if any.
3. The meaning of the data (i.e., a description).
4. A mathematical symbol to represent the data, if appropriate.
5. The type of data (e.g., floating-point, integer, or character).

A self-describing file protocol (SDFP) should incorporate these attributes automatically. Further, the user of self-describing data obtains the data by name. That is, unlike traditional text and unformatted data, SDFP does not require the user to know, for example, the column of a table in which a certain quantity appears.

This is a crucial feature of the self-describing data concept, since it enables one to construct generic programs that can perform similar operations in several applications. A good example would be a graphics program that plotted data by name, rather than being written specifically for a single application. (Compare this with the current state in accelerator physics, where many similar simulation codes have their own custom-made graphics packages.)

For the purpose of discussion, assume that the data can be meaningfully organized into a single table. This table might, for example, contain various quantities such as position, Twiss parameters, element name, etc. Any program compliant with the SDFP in which the data was written would be able to access the columns of this table without regard as to how the data was organized. The program would access the data by name, using routines supplied by the creator of the SDFP. If the program needed only certain columns of data (e.g., position and horizontal beta function) it would access only those columns. The presence of additional columns of data (e.g., dispersion), would be irrelevant. Furthermore, the program would not need to know the source of the data ; it could be from any source, including direct user input into a file, output from any compliant simulation code, or measurement on a real accelerator.

[1]Work supported by U.S. Department of Energy, Office of Basic Energy Sciences under Contract No. W-31-109-ENG-38.

There are many accelerator codes that perform comparable computations and produce comparable output in dissimilar formats. If all of these codes employed the same SDFP for their output, users and programs could access data without regard for the code that generated it. Post-processing or multistage simulation would be possible without custom integration of programs. That is, programs would not need to be custom designed to provide output to each other in order to work well together. This would happen automatically through the SDFP mechanism. This would allow users to combine programs in ways that were not planned by the creators of the programs.

The only constraint would be that the codes used the appropriate names for quantities. In practice, this restriction can be reduced by providing a name translation table. Programs can also be designed to request data under several different but equivalent names, e.g., "betax" and "betah" for the horizontal beta function.

Another significant advantage of using an SDFP is the possibility of making program "toolkits." We have already noted that using an SDFP allows programs to read data from many sources. If a program is created that both reads and writes data in the same SDFP, then this program may potentially be used to alter data "upstream" of any other program that reads the SDFP. If many programs exist that fulfill this requirement, then these programs can be employed in nearly arbitrary sequences to process data. The usefulness of each program is vastly increased by the possibility of using it as part of such a sequence. Such a set of programs comprises a toolkit, taking advantage of the SDFP to virtually guarantee that any member program will provide readable data to any other member program.

2 The SDDS File Protocol

The principles elucidated in the previous section have been implemented in the Self-Describing Data Set (SDDS) protocol, developed at APS. Any SDFP implementation must make certain assumptions about what type of data will be stored and how it will be arranged. These assumptions comprise the data model for the protocol.

An SDDS file consists of a header section and a data section. The header section declares that the file is an SDDS file and which SDDS version it is in. It further defines zero or more data elements that together constitute a data structure. The SDDS model organizes the data section into a series of "data pages," each of which is an instance of the data structure defined in the header. That is, each page of any file must contain the same elements but may contain different specific data. For example, each page could contain the Twiss parameters for a different accelerator or for a different tuning of the same accelerator.

Within each page, the following classes of data are recognized:

1. Parameter data, consisting of single values that may either be fixed throughout the file or vary from page to page.
2. Tabular data, consisting of an arbitrary number of rows of mixed-type data. The data in the table is referred to by the name of the column. Any number of columns may be defined, but the same columns are expected for each page of the file.
3. Array data, where each element is of fixed but arbitrary dimension (i.e., an arbitrary number of array indices is allowed). Each array is accessed separately, by name, but may be placed in a group with other arrays. The size of an array may vary from page to page. (Note that a parameter is essentially an array containing a single value, but it has simplified access from within a program compared to an array.)

Any element of these data classes may have one of the following C-language data types: `float`, `double`, `short`, `long`, `char`, and `char *`. These are, respectively, singleand double-precision floating point, short and long integers, single characters, and character strings. All of these types may be mixed in the tabular data section, but each column must contain data of fixed type. SDDS has no restrictions on the numbers of each type of element, on the length of the tabular data, on the dimension of arrays, or on the size of arrays.

To continue the Twiss parameters example, one might create an SDDS file containing:

1. Parameters: The name of the lattice, the tunes, the chromaticities, the acceptances, etc.
2. Columns: The element name, the position, the Twiss parameters, apertures, etc.
3. Arrays: The response matrix, matrices for tune and chromaticity adjustment, etc.

While it is possible to design a self-describing file protocol that is more general than SDDS, we have found that SDDS allows us to conveniently store almost any data. At worst, the user may need to store disparate data in different, parallel files; this actually has the advantage of making the data easier and faster to access. When evaluating the desirability of a more flexible data model, it is important to recognize that too much generality will adversely affect the ease of development, ease of use, and development time. The complexity of highly general models may explain why they are not in wide-spread use.

Some examples of the types of data stored in SDDS at APS are: particle tracking input and output; Twiss parameters and transfer matrices along an accelerator; beam loss data from tracking simulations; simulated orbit/trajectory correction results and statistics; orbit/trajectory data from the APS accelerators; simulated rf cavity fields; backup/restore files for the APS accelerators; archival data for the APS accelerators; video images and other density maps; data transferred from digital oscilloscopes and spectrum analyzers.

Of course, each of these types of data could have been stored in other types of files. What is new is that SDDS allows the user to store all of these types of data in the same type of file and to use the same set of tools with all of them. For example, the same program that plots beta functions versus position can also be used to plot archived oscilloscope signals versus time, or to make scatter plots of simulation tracking data. Similarly, the same program that does FFTs of simulation tracking data from a simulation can do FFTs of turn-by-turn BPM readings from the APS storage ring. These FFTs, of course, would be plotted by the same program as all the other data.

Another advantage of SDDS is that the data may be either ASCII or unformatted (i.e., "binary"). The SDDS header is in ASCII and has a familiar namelist format. It is a simple matter to create an SDDS file using print statements within a program, giving immediate access to a range of SDDS tools (see the next section). The SDDS header also has features to make it easy to convert existing text data into SDDS protocol; in many cases, one simply creates a header and attaches it to the top of the file. We have found that these features were extremely important in helping users to switch from text printouts and other custom formats to SDDS. As noted above, SDDS incorporates a protocol and version identification string as part of the header. This allows immediate detection of whether a given file is in SDDS protocol, and what version of the protocol it is using. The SDDS library routines are configured to allow reading of any SDDS version, so that an SDDS data file never becomes obsolete. This permits upgrades of the protocol itself without disrupting users.

3 The SDDS Toolkit

The SDFP concept is a break with the traditional way that accelerator physicists store data. A further break is the introduction of the toolkit concept for data processing. As described above, a toolkit is group of independent but cooperative commandline programs. Traditionally, data processing has involved writing single- or few-purpose programs that use data from a single source (e.g., a particular simulation or experiment). This is in spite of the fact that the operations involved in different data processing applications are often essentially identical except for the specific data involved. With the use of an SDFP, it makes more sense to write generic programs that are not specific to experiments or simulations. This is because an SDFP allows one to access data by name, which permits symbolic specification of operations. The simplest example conceptually is graphics, as described briefly in the previous section; there is no good reason that a single graphics program cannot be used for almost all of the codes and data employed in accelerator physics. (In fact, there are several simulation codes in use at APS that use only SDDS files [1].)

While toolkit programs can be shared by users, development is decentralized. Since the only requirement for a toolkit program is that it read or write SDDS files, anyone may make a contribution without any negative impact on the rest of the toolkit. This is an advantage over "all- in-one" packages, which force the user to import his data into a single program and perform all operations within a centrally-controlled environment. In contrast, toolkit programs are combined through use of the local command shell and through command scripts [2]. SDDS-compliant programs automatically work together, with little or no planning on the part of code developers. To date, the authors know of nine individuals at APS who have developed SDDS- compliant programs.

The SDDS Toolkit is a growing group of about 60 programs that uses SDDS files. Most of the programs both accept SDDS input and produce SDDS output. The following sections provide a brief description of a number of the programs, grouped by functional type. Several programs are given more detailed descriptions. The reader may note that most of the programs process column data only, producing either new columns or parameters that reflect the processing. This is an indication of the extent to which this single feature of the SDDS data model serves almost all the cases we have encountered.

The reader is referred to our companion paper [2] for a list of some of the APS commissioning data that has been processed using the SDDS Toolkit and examples of how the Toolkit programs are coordinated. Extensive hypertext manuals are available with the SDDS distribution package for the Toolkit itself [3] and the SDDS-compliant EPICS programs [4]. A tutorial introduction with an emphasis on EPICS applications is available in [5].

4 SDDS Toolkit Programs

The intention is of this section is to provide an overview of some of the SDDS Toolkit programs that are presently available. For most programs, a brief and usually very incomplete description is given. For the less obvious programs, a brief example is given. A few of the most heavily-used programs are given long descriptions.

4.1 Data Analysis Tools

- `sddschanges`: Analyzes changes in column data from page to page in a file, relative to a reference file or the first page. An example application is computing changes in waveform data over time relative to an initial waveform.
- `sddscorrelate`: Computes correlation coefficients and correlation significance among multiple columns of data. An example application is searching for possible causal relations among many columns of time-series data.
- `sddsderiv`, `sddsinteg`: Numerical differentiation or integration of multiple data columns versus a single column.
- `sddsdigfilter`, `sddsfdfilter`: Perform time- and frequency-domain digital filtering.
- `sddsenvelope`: Analyzes column data across pages to find minima, maxima, averages, standard-deviations, etc. on a row-by-row basis. An example application would be finding the envelope and average of a repeatedly-sampled waveform.
- `sddsexpfit`, `sddsgfit`: Exponential fit or Gaussian fit to column data.
- `sddsfft`: Computes fast Fourier transforms and power spectral densities of column data. Will provide real and imaginary parts, magnitudes and phases.
- `sddshist`, `sddshist2d`: One- and two-dimensional histograms and simple statistics.
- `sddsinterp`: Does arbitrary-order polynomial interpolation of multiple data columns as a function of a single column.

- `sddsoutlier`: Eliminates statistical outliers from data using, for example, a limit on the number of standard deviations by which a point may differ from the mean of the group.
- `sddspeakfind`: Finds values of many columns at locations of peaks in a single column.
- `sddspfit`: Does arbitrary order polynomial fits to column data, including error propagation and adaptive fitting.
- `sddsprocess`: A generic data processing program that, among other features, permits: defining new tabular and parameter data in terms of existing data using user-specified expressions; printing, scanning, editing, and subprocess execution (with output capture) of string data; units conversion; selection of tabular data rows by multiple numerical windows and wildcard matching, with user-specified logic among selection criteria; creation of parameters based on any of 29 types of analyses of tabular data.

  The tabular data analyses include statistics, waveform and pulse parameters and linear fit parameters. Among the statistics are arithmetic average, rms, standard deviation, median, minimum and maximum. Waveform analyses include amplitude, baseline, risetime and falltime. Analyses may be invoked for an arbitrary number of data columns using wildcards, with parameter names being constructed from a user-supplied template. Many analyses support weighting and winnowing based on data from secondary columns.
- `sddspseudoinverse`: Reads the numerical tabular data as though it was a matrix and outputs the SVD inverse. The tabular data need not form a square matrix. Options allow one to adjust the number of singular values selected. This program is used to calculate gain matrices in a number of generalized feedback applications.
- `sddssmooth`: Smooths columns of data using multipass nearest-neighbor averaging and optional "despiking." Often used in concert with `sddspeakfind` to give reliable peak frequencies from noisy spectra.
- `sddsslopes`: Calculates the slopes and intercepts of selected columns using another column as independent variable. When applied to data logged at intervals using the time data column as the independent variable, this program directly produces time trends. This program has also been used extensively in calculating linear responses of systems to actuators. As such, `sddsslopes` and `sddspseudoinverse` are often used to analyze experimental data to produce a gain matrix for feedback.

4.2 Data Manipulation Tools

- `sddsbreak`: Breaks data pages into new, separate pages based on changes in column data and other criteria. For example, `sddsbreak` can analyze time-series data to find probable time gaps and make a separate page for each contiguous section.
- `sddscollapse`: Collapses a data set into a single data page by deleting the tabular data and turning the parameters into columns. This is an important tool in data collation operations. Typically, one processes multipage tabular data to produce parameters for each page (e.g., statistics for several columns), then collapses the result to obtain a single page of tabular data. Several such data sets can be combined using `sddscombine`, which puts one formally in the same position as when one started. This program is often used with `sddsprocess` in data reduction cases where thousands or even millions of data points must be reduced to a simple result.
- `sddscombine`: Combines any number of data sets into a single data set by adding data from each successive data set to a newly-created data set.
- `sddsconvert`: Allows conversion of a data set between binary and ASCII, with optional deletion and renaming of columns, arrays and parameters.

- sddssort: Sorts the tabular data section of a data set by the values in any number of named columns and optionally eliminates duplicate rows.
- sddstranspose: Reads the numerical tabular data as though it was a matrix and outputs the transpose. Applying sddstranpose again to the output reproduces the original file. This program is used often to convert multi-column one-row SDDS data sets, obtained from sddscollapse, say, into two-column multi-row data sets (an extra string data column comes from the original column names), which are then suitable for plotting. For example, sddsprocess and sddstranspose could be used to take a file with 360 columns of beam- position-monitor samples and create a plot of rms position variation vs BPM name.
- sddsxref: Creates a new data set by adding selected rows from one data set to another data set. The rows are selected by matching the string or numeric values in a specified column that is present in both of two pre-existing data sets. Will also transfer parameter and array data between files.

4.3 Graphics Tools

- sddscontour: Makes contour and color-map plots from a data column interpreted as a matrix, or from many data columns. Optionally does FFT- based interpolation and filtering. Uses the same device drivers as sddsplot.
- sddsplot: A very powerful generic device-independent graphics program for plotting tabular and parameter data. The program is equally capable of "quick-and-dirty" and publication-quality plots. Among the supported devices are X-windows, color and black-and-white Postscript, Tektronix, REGIS, and various PC graphics cards. While the program is commandline-driven, it brings up a GUI under X-windows. (Support for graphics using such old methods as the Tektronix and REGIS languages may seem unimportant; however it does enable one to get graphical displays using a wide variety of terminal emulators running on personal computers.) sddsplot was designed to handle and organize large amounts of data. Any number of data elements may be plotted from any number of files. The data to be plotted from any file is simply named on the commandline, with optional wildcarding. The program organizes data into a potentially unlimited number of "panels" on a potentially unlimited number of logical output pages; in this context, a panel is a stand-alone graph containing zero or more data points from any number of sources and occupying all or part of a page. When desired, grouping of data into panels may be specified in numerous ways, such as grouping columns of data by tag parameter values stored in the data files, or grouping columns of data from many different files by the name of the data. For X-windows, a movie mode is available that displays pages in rapid succession. Figure 1 shows an example of a page from a movie made from tracking data for the APS booster; the overlayed histograms are made with sddshist. Figure 2 shows an example of plotting Twiss parameters using sddsplot.

  Plotting types include lines, symbols, error bars, dots, impulses or bars and arrows; things like color and symbol type may be varied automatically by the program to distinguish different data, or these attributes may be specified explicitly. When plotting lines, sddsplot can optionally analyze data for gaps and break the line at appropriate points. Data may be filtered and subjected to matching operations based on data in other columns or parameters, as well as sparsed and randomly sampled. Title, label, and legend strings for each set of data may be extracted from the corresponding SDDS file.

4.4 Miscellaneous Tools

- File Protocol Convertors: Programs are available to convert from the following into SDDS: Hewlett-Packard CITI; Hewlett-Packard HP54542 scope format; Spiricon Laser Beam Analyzer and others.

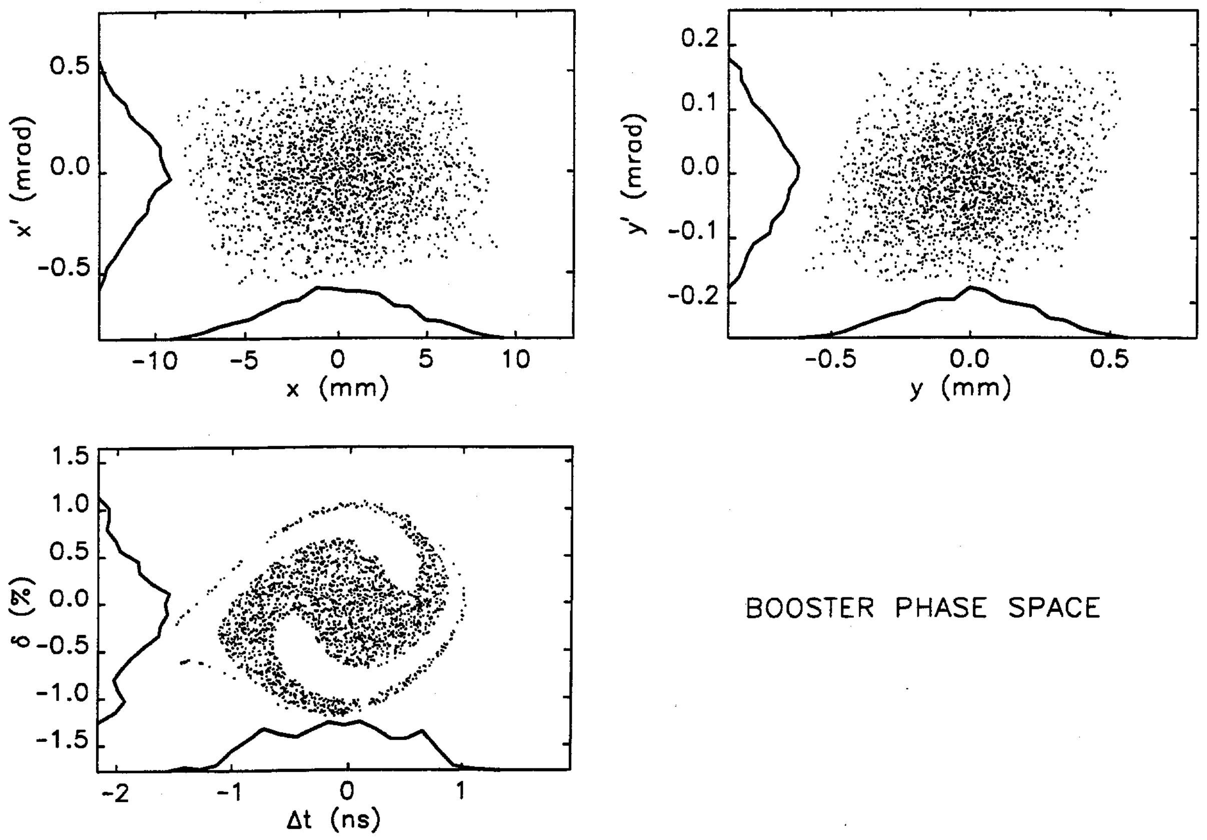


Figure 1: A frame from a movie of APS booster 6-D phase-space tracking simulation, with a deliberate mismatch.

Programs are available to convert SDDS to the following: Mathematica[2]; Excel and Wingz spreadsheets and others.

- `sddsprintout`: Makes customized printouts from SDDS data. This means that while an SDDS file may contain dozens or even thousands of parameters and columns, a printout can be easily made that contains just the data of interest.
- `sdds2stream`: Takes column or parameter data from a list of SDDS data sets and delivers it to the standard output as a stream of values. Often used to get data into a shell variable in `csh` or `tclsh`.
- `sddsquery`: Prints a summary of the SDDS header for a data set, or delivers this data as a new SDDS file.

5 SDDS-Compliant EPICS Tools

The control system at APS is the Experimental Physics and Industrial Control System (EPICS) [6]. A toolkit approach is adopted in designing workstation applications that communicate with EPICS. In order to take advantage of the existing SDDS toolkit, those applications that exchange data between EPICS and data files use the SDDS file protocol.

The tools listed below are very general. They do not assume that an accelerator is being controlled. These tools can easily be adapted to non- EPICS control systems by replacing the EPICS channel access libray calls in the source codes with appropriate substitutes. A set of detailed case studies of the use of some of these tools is available in [2].

[2]Mathematica is a registered trademark of Wolfram Research, Inc.

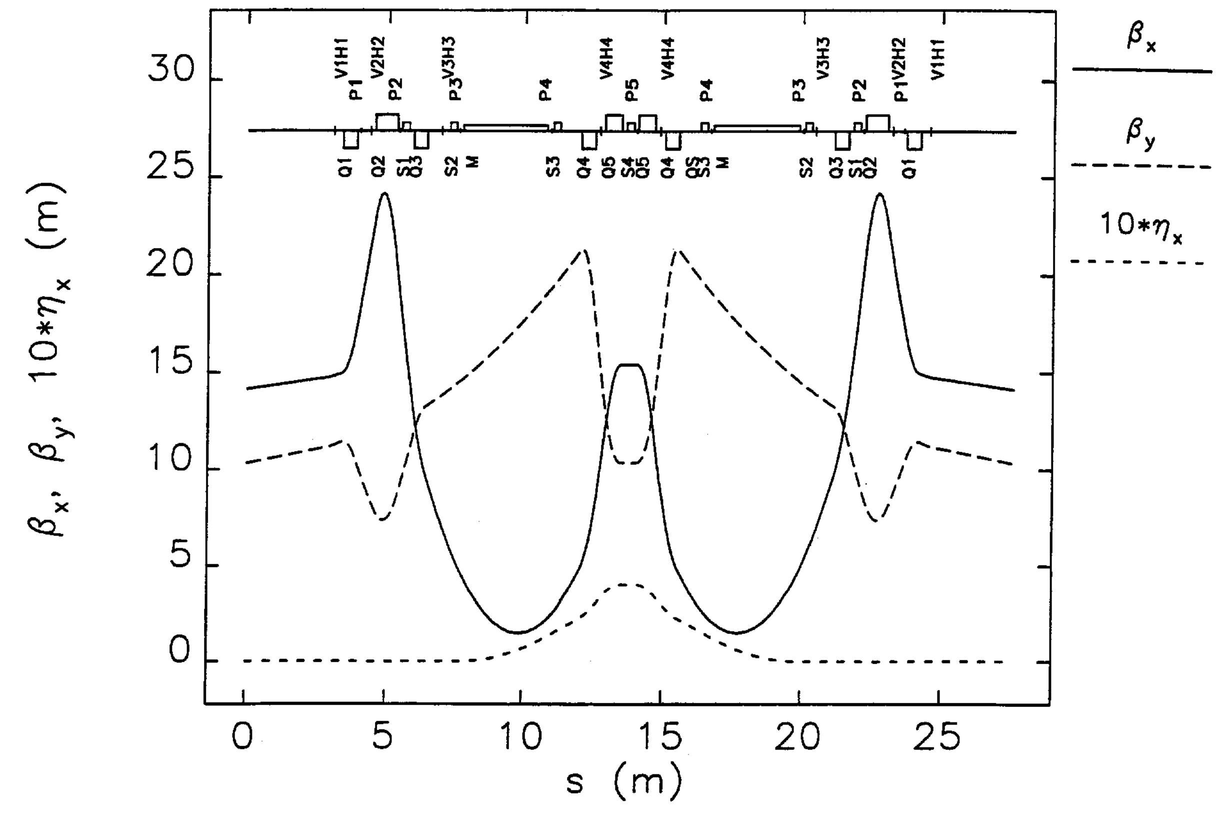


Figure 2: Twiss parameters for one sector of the APS ring.

5.1 Configuration Save and Restore Tools

- `burtrb`, `burtwb` (Back-Up and Restore Tool): These programs perform EPICS save and restore operations, reading data from and writing data to the control system. SDDS protocol is used both for the "request file" (configuring a read operation) and the "snapshot file" (containing data from a read operation or for a write operation). Because SDDS is used, a snapshot file can function as a request file, as it contains all of the essential data from the request file. Information such as time stamp, user ID and operator comments is stored in the file as SDDS parameters; additional parameters may be added at will without side effects. Operations such as selection of subsets and comparison of snapshots are performed using various SDDS Toolkit programs, forming the basis for a save/compare/restore facility [7].

- `toggle`: Alternates between two sets of process variable values stored in snapshot files at a regular interval. Example application: Alternate between two configurations of an accelerator for debugging accelerator performance.

5.2 Data Collecting Tools

- `sddsmonitor`, `sddsstatmon`, `sddsvmonitor`, `sddswmonitor`: These programs provide various types of data logging into SDDS files. Each is configured by one or more SDDS files containing names of process variables (PVs) to read. The various versions are, respectively, for logging values for simple lists of PVs; statistics for the same; values based on list multiplication and organized into vectors and waveform PVs. The vector and waveform versions will accept input data for scalar logging as well, thus partially duplicating the function of `sddsmonitor`. Normally, the programs log data at a fixed, user-specified time interval. However, options are provided for event-based data logging. For all of the programs, data taking may be controlled by a parent process using messages in the standard input stream. All except sddsstatmon accept additional SDDS input specifying test limits for conditional data logging. For sddsmonitor, data may be stored in a circular buffer and written to disk only when user-defined trigger or glitch events are detected.

- sddssnapshot: Reads values of process variables and writes them to a snapshot file. This program differs from burtrb in that it may operate in a server mode in which a new file is written to a named output file whenever the signal SIGUSR1 is received. An application would be taking successive sets of data triggered on certain events determined by another process, without remaking any channel access connections.

- sddsexperiment, sddsvexperiment: These programs provide for experiment execution and data collection. The most basic usage involves variation of a setpoint in equal steps, with data gathering for each setpoint. More sophisticated applications may involve variation of an arbitrary number of setpoints on a multidimensional grid. Also supported are computation of setpoints from user- supplied equations, allowing for nonlinear variation. Subprocess execution is supported to permit scripts to be run in coördination with setpoint variation and data gathering; these scripts may acquire specialized types of data (e.g. video images) or perform specialized tasks (e.g. adjusting a motorized attenuator to obtain a specified power level). Data gathering includes optional computation of averages and error bars at each measurement point. (The two variants of the program differ in the mechanism by which the measured process variable names are supplied and the way the data is organized in the output file.)

5.3 Control Tools

- sddcontrollaw: This program performs simple feedback on process variables. Basically, a set of control process variables is used to regulate to zero a set of readback process variables. Optionally, the program can hold constant the initial values of the readback process variables. The values of the control process variables ($\mathbf{u}_n$) applied at some iteration of the feedback are related to the values of the readback process variables ($\mathbf{x}_n$) at the previous iteration through the matrix equation $\mathbf{u}_n = \mathbf{u}_{n-1} - K\mathbf{x}_n$, a variant of the control law equation in control theory. An SDDS input file specifies the gain matrix K and all the process variable names.

  To make the control robust, a series of validity tests on process variable values are implemented by means of an additional SDDS file containing the names of process variables and their corresponding limit values. Feedback iterations proceed only when all of the test process variables are within the specified range. When one of the tests fails, control is suspended (i.e. the control variable values are untouched by the program) until all of the tests succeed.

  Some of the applications have been: removing slow energy drifts in the linac, correction of linac and most beamline trajectories, correction of positron accumulator ring and storage ring orbit, "slow" feedback on storage ring gap voltage amplitude to compensate for beam loading and performing complex experiments in association with sddsexperiment. The gain matrix K is usually determined empirically with the use of sddsexperiment or sddsvexperiment (to determine parts of the response matrix) and other SDDS tools (to invert the response matrix to obtain the gain matrix). The process of determining the gain matrix and applying sddscontrollaw for the different systems of the APS accelerators is essentially the same, only the names of process variables are changed.

- squishPVs: This program provides for automated "knob tweaking." It was originally written for first-turn trajectory correction in the APS ring, but is of more general application. The name comes from the way the program flattens out the trajectory. The program takes an SDDS input file giving a series of actuator names and any number of readback names for each actuator. It then adjusts each actuator in turn to minimize either the rms or mean-absolute-value of the corresponding readbacks. While quite slow compared to the use of a response matrix and `sddscontrollaw`, `squishPVs` has the advantage of being completely insensitive to, for example, magnet and beam-position-monitor polarity errors.

6 Acknowledgements

The authors wish to acknowledge the following individuals for contributions to the SDDS toolkit and EPICS-specific tools, either in the form of suggestions, bug discoveries, or contributed programs: J. Carwardine (APS), Y. Chung (APS), K. Evans (APS), N. Karonis (FNAL), E. Lessner (APS), S. Milton (APS), G. Rinehart (IPNS), C. Saunders (APS), M. White (APS).

References

[1] M. Borland, "A Self-Describing File Protocol for Simulation Integration and Shared Postprocessors," Proceedings of the 1995 Particle Accelerator Conference, May 1-5, 1995, Dallas, Texas (to be published).

[2] M. Borland, L. Emery, N. Sereno, "Doing Accelerator Physics Using SDDS, UNIX, and EPICS," these proceedings.

[3] M. Borland, "User's Guide for SDDS Toolkit Version 1.4," http://www.aps.anl.gov/asd/oag/manuals/SDDStoolkit/SDDStoolkit.html.

[4] L. Emery, private communication.

[5] J. A. Carwardine, "An Introduction to Plant Monitoring Through the EPICS Control System," these proceedings.

[6] L. R. Dalesio, M. R. Kramer, A. J. Kozubal, "EPICS Architecture," in *ICALEPCS* 1991, pp. 278–281.

[7] D. J. Ciarlette, R. Gerig, "Operational Experience from a Large EPICS-Based Accelerator Facility," these proceedings.

MIGRATING THE CERN PS CONTROL SYSTEM TO IBM WORKSTATIONS

Nicolas de METZ-NOBLAT
PS Division, CERN, CH-1211 Geneva 23, Switzerland

ABSTRACT

The workstations used within the control system of the CERN PS accelerator complex are not produced any more. We had therefore to review the software primary used as user interface and we made a port to IBM workstations.

We are also preparing the maintenance of this code for the next ten years with minimal staff. This implies a clear separation between general computing facilities, control system developments, and operation.

In order to share our experience, we will try to summarize various aspects of this migration:

- system installation principles used to speed-up error recovery time and reduce long-term maintenance costs,
- problems resulting from the coexistence of two different platforms during migration,
- software problems due to the platform and operating system changes,
- hidden dependencies on a specific manufacturer.

1. INTRODUCTION

During the last five years, the PS Division has carried out a large project to renew the control system of the CERN PS particle accelerator complex [Ref. 1 & 2].

This resulted in the introduction of a rather large number of UNIX workstations and X-terminals mainly used for the daily operation and the exploitation of the CERN PS complex.

Our original systemused DEC workstations based on 32 bit MIPS microprocessors. The new generation of DEC workstations is based on a new 64 bit chip and a new operating system. The amount of work required to port our control software to this new generation was almost the same as that necessary to port it to any other UNIX workstation. After detailed cost analysis, we decided on IBM RS/6000 workstations based on PowerPC chips and running the AIX operating system.

The introduction of a new generation of workstations is also a good time to review organizational details in order to minimize the staff required for the long-term maintenance of such computing resources.

2. NETWORK TOPOLOGY [Fig. 1]

We have been introducing a large number of workstations (>100) and diskless front-end computers (>100), initially to develop a new control system and then to operate and maintain it. The problem was to keep this maintainable.

We split the equipment into two different networks isolated by a router:

- An office network: This covers development and test equipment
- A control network: This concentrates the equipment needed for normal operation of the complex

Then we split this control network into various subnets, at present isolated by bridges:

- a central control subnet on which we keep some critical services like central timing generator or databases,
- one subnet for each accelerator (or group of accelerators):

- one for the proton and the ions LINACs,
- one for the LEP Pre-Injector complex (LIL and EPA),
- one for the PS Booster (PSB),
- one for the CERN Proton-Synchrotron itself and its transfer lines (CPS).

We organized each subnet to still allow a minimum of operation from a local control room, even when disconnected from the central control subnet. Diskless computers rely on a single file server located on the same subnet to recover after a power-cut.

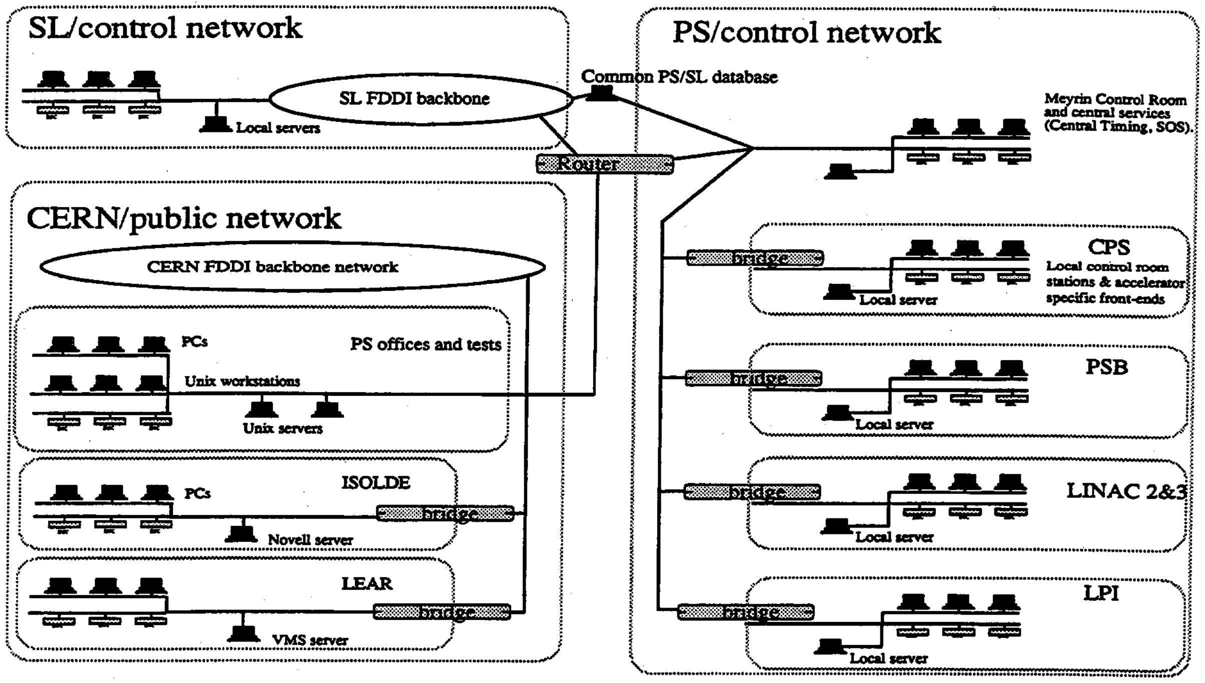


Fig 1: The network topology

3. FILE SYSTEMS ORGANIZATION

We had to decide early on a well-structured file organization that could stay stable during the whole project.

We decided on the following organization:

- One file system per accelerator: This receives all programs required for the minimum operation and all software specific to this accelerator, including software running in diskless front-ends.
- One file system for central services: This receives programs that are not critical for the minimum operation and used for the control of more than one accelerator..
- One file system for the control software source repository: this receives sources of all the programs developed for the purpose of the accelerator control. Its structure reflects the target file systems in order to ease long-term software maintenance.
- One file system to receive libraries and associated include files for each different target. This file system also receives specific programs required either to produce the applications or to maintain operational data.
- One file system to receive Word-Wide Web documentation used for on-line help.

The CERN central computing service furnishes other resources not used for normal operation:

- Home directories can be located on central servers,
- Public domain software must continue to be maintained centrally,
- Whenever possible, commercial products must also be centrally maintained.

4. MIGRATION STRATEGY

Before deciding on new UNIX workstations, we did a preliminary inventory of all our software and our dependence on specific features. Nearly 30 different programmers had developed about 100 applications. Then, we ported a significant application to various platforms and evaluated the constraints.

The next step was to start the porting with a small number of workstations and servers in order to prepare a validation of almost all applications in a real context co-existing with the current environment [Ref. 3].

We had to minimize the staff required and interference with on-going developments and normal beam production. We prepared an independent environment to receive sources and libraries and a clone of existing operational file systems. This allowed us to prepare for the final software distribution and to execute applications in a realistic environment, keeping the same data source database for software customizations such as program menus and equipment lists.

The next step (March 1996) will reverse the situation between old and new workstations in order to avoid the simultaneous maintenance of two binaries for the same applications.

The final step will be to reduce the platform diversity in order to reduce system management complexity and to ease new application development.

5. PROBLEMS ENCOUNTERED DURING THE MIGRATION

While developing our applications, we tried to avoid the use of manufacturer-specific extensions to standards and the migration was a good occasion to verify this point.

We did not encounter any major problem while porting almost all the software to the new platform.

Some problems originated from librarys updates for such as standard Motif and X11, or from our own libraries. With a new development environment on both platforms, it was possible to isolate and cleanly these problems.

Other problems were due to dependencies on the environment:

- Some programs were using specific fonts.
- Simultaneous use of various UNIX variant compatibility routines (Berkeley Software Distribution, System V and POSIX) does not always mix properly.
- Our front-ends already had a different byte order and most programs were already ready for such a change.
- C compilers and system include files differ, but without any major incompatibility.
- We tried to avoid as most as possible the use of Display Postscript extension, except for two applications.
- One application used a specific external video input extension and multimedia libraries. We will need to adapt it and hope that with current developments in the multimedia area some standards will be available for this field.

The migration itself was an occasion to locate small programming mistakes, or surprising limitations introduced by the programmer. A typical example was the limitation to six characters of a host name!

We also had to prepare the future development environment. One example is that we had to decide on a new User Interface Builder program.

6 OTHERS SYSTEM CONSIDERATIONS

We are trying to prepare long-term exploitation solutions:

- We try to use the same system installation procedures as other CERN central UNIX services with minimum specific customization, in order to share experience and support.
- We prepare all critical computers for remote exploitation. This applies to servers and to front-ends.
- Our network layout is clean, easy to understand and is organized to allow longer intervention delays on non-critical equipment.
- We keep coherent contents within file systems and have automated backups.

7. CONCLUSION

This exercise of porting our control software was very positive. It was an occasion to evaluate in details our dependencies, and to check its completeness. It demonstrated the ability of a UNIX system to maintain a large amount of software written in C. However, it is very important to choose emerging standards and to avoid any specific extensions to these.

During the next year we expect to reach a very clean and stable situation with workstations, servers and control application software. That might open the way to some out-sourcing solutions of our problems as we have already experienced with the system management of the LEAR control system.

ACKNOWLEDGMENTS

I would like to thank Michel Arruat, Franck di Maio, Serguei Moukhine, Orel Oleg and Fabien Perriollat for their great help in achieving this work within the planned timetable. I would like also to thanks all applications developers for their collaboration and the various people who have been testing the resulting environment.

REFERENCES

[1] F.Perriollat, Ch.Serre, “The new CERN PS control system, Overview and status”, ICALEPCS, Berlin, Germany, October 18-23, 1993, Nucl. Instr. And Meth. A352 (1994) 86.

[2] R.Rausch, Ch.Serre (editors), "Common Control System for the CERN Accelerators", Proceedings ICALEPCS, Tsukuba, Japan, 11-15 November 1991, p.54-58
The PS and SL controls groups, "PS/SL Controls Consolidation Project", Technical Report, CERN PS/91-09(CO), CERN SL/91-12(CO), April 91.

[3] M.Boutheon, F.DiMaio, A.Pace, "General Man-Machine Interface used in Accelerator Controls; Some Applications in CERN-PS Control System Rejuvenation", Proceedings ICALEPCS, Tsukuba, Japan, 11-15 November 1991, p.452-455

Migrating From X-Window Workstations To Windows NT PCs

David E. EISERT
Synchrotron Radiation Center, University of Wisconsin-Madison, 3731 Schneider Drive, Stoughton, WI 53589-3097, USA

The operator interface for the synchrotron storage ring Aladdin has evolved many times over the years to keep pace with new technology. Recently we have started a migration from VaxStation 3100s running VAX/VMS to PC clones running Windows NT. Although we could upgrade the existing workstations with the latest model, there are other reasons to change operating systems. The most important reason is the ability to integrate commercial PC applications such as LabView from National Instruments into the existing control system. In addition, the software development packages available under Windows NT are integrated more closely with the windowing environment than their VAX/VMS counterparts. Some of the advantages and disadvantages we have uncovered in porting existing code and in integrating commercial PC applications are discussed.

1. INTRODUCTION

The first version of the operator interface for the Aladdin control systems was implemented on color graphics terminals and a minicomputer [1]. As we acquired graphics workstations the interface remained largely unchanged for several reasons. Firstly, the operation staff were familiar with the existing interface and were reluctant to lose some of the features of our non-standard graphical user interface (GUI). Another problem developed due to the evolution of GUI environments; it was easier not to use any of the new features than to keep up with the fast changes in these environments. In addition, when we made our last major revision of the operator interface code, the workstations lacked graphical software development tools.

Over the years we have made some improvements to our operator interface software to increase efficiency when porting to a new GUI environment. The software has been consolidated from many medium sized applications to one large application and five small applications. Only two applications utilize the graphical display, the other applications perform various monitoring and file recording tasks in the background. The original software was written in FORTRAN and it was difficult to interface with GUI environments written in C. During the migration to X Windows the original FORTRAN code was rewritten in C. In addition, the parts of the software that use the display have been divided from sections that perform calculations and I/O operations.

There are several reasons to move from X Windows to Windows NT. Our staff has almost entirely converted over from the VAX/VMS systems to the PC platform. They wanted the ability to take work home and use the same computers at home as they do at work. They also wanted access to the many user-friendly PC applications. Afterwards they noticed that some of the PC applications would be very useful for machine studies and data manipulation. Another reason to switch platforms is the cost of color workstations compared to PC clones. The differences in the initial purchase price are narrowing rapidly but workstations will always be more expensive. The workstations also require a service contract since we can not afford to purchase spare workstations. Spare PC clones are available in emergency situations from almost every desktop and spare parts are available from local retailers.

2. DEVELOPMENT TOOLS

VAX/VMS has been a leading platform for software development for many years. The compilers have always been regarded as generating extremely efficient code. Unfortunately the development tools available under VMS never made the transition to the GUI environment. Software development tools always use a command line environment with each tool acting independently of the others. Compiler vendors created the Integrated Development Environment (IDE) under Windows/Windows NT. Many Windows compilers do not require the use of the IDE environment and at first people familiar with a command line environment will probably avoid using it. Eventually it becomes apparent that there are many advantages to working in an environment where all the development tools work together in one package.

2.1 Compilers

C++ compilers have been available for a long time on the PC platform but have only recently been offered by DEC. VAX C did not implement very strict type checking and allowed many practices that generate warning messages in current C++ compilers. More recently we started using the DEC C++ compiler but very few of the warning conditions were corrected since it has an option to implement the VAX C style type checking. When moving to a PC C++ compiler much of the type checking had to be turned off to avoid hundreds of warning messages.

Fortunately the PC C++ compilers have many options when it comes to warning messages, errors and optimizations. These options could be controlled by command line style switches but the IDE makes setting these switches easy through the use of a dialog box. Due to the IDE the whole development environment has many more options than is typically available with command line environments. In addition, GUI code normally has very large include files but PC C++ compilers precompile these include files. The next time a module is compiled, the include files do not have to be processed.

2.2 Debuggers

There are basically two stages of development for debugging software. During the first stage of debugging one would build the code with all the information needed for a source code debugger. The debugger is normally invoked and the code will run normally unless an error is encountered. When an error is encountered the debugger halts the execution of the code and enters the debugger. After the code is initially debugged a second release version of the code is ordinarily produced. The release version does not have all the information needed for a source level debugger. When a bug is encountered the system may produce one of several results. The worst case is that the system crashes without returning any information about the problem. In the best case only the program terminates and enough information is returned before it terminates to identify and correct the problem.

Both systems are fairly comparable when running under a debugger. The VAX/VMS Debugger supports a GUI-based debugger but lacks many features available in Windows NT. The VAX/VMS Debugger is primarily a command line debugger with an add on GUI interface. As a result the Windows NT version is a better match with the GUI interface. For example, the executing code is highlighted in the source code editor rather than just displayed in a special debugger window. The Windows NT version also allows the debugger to be invoked after an error is encountered. Normally the debugger has to be invoked before the program is started but under NT if an error is encountered the debugger can be started at the point the error occurred.

The early MS-DOS/Windows operating system lacked any memory protection from the released version of an application program. Normally every programming error encountered would produce an entire system crash with little information about the cause of the error. Windows NT has an extensive number of protection mechanisms for the operating system. Earlier windows code that took advantage of the lack of memory protection will fail under Windows NT. When a program running under Windows NT encounters a programming error a dialog box pops up explaining there was a fatal error. Some information on the location of the error can be obtained from the information reported in the dialog box. Unfortunately the Microsoft Visual C++ compiler normally does not include procedure names in the executable image, therefore only the address of the error can be reported. It should be noted that we have never crashed the Windows NT operating system due to an application programming error. Windows NT is able to recover all resources used by the errant program including any networking resources. VAX/VMS systems produce a program stack dump on fatal program errors with the procedure names and specific address information. VAX/VMS provides a slight advantage over Windows NT by providing the names of the procedures where the error occurred.

2.3 Object Editors

Early efforts to develop GUI code encountered difficulties when laying out the elements of a dialog box. The origin pixel coordinates and pixel dimensions of the elements had to be entered by hand in a text file. Later development tools were supplied that allowed graphical editing of the elements in dialog boxes. Both X Windows and Windows NT have similar utilities for creating and editing the GUI visual elements. Although the Windows NT GUI editor is built into the development package IDE interface, the X Windows editor is available only as a separate tool.

2.3 Other Tools

Some other tools that are useful during the software development process include intelligent program editors, software version control and code performance profilers. The early editors supplied basic text editing features such as text entry, navigation through text files, text searching and replacement. Later versions like the VAX/VMS Language Sensitive Editor included enhancements for checking code syntax prior to compilation. Editors available under Windows NT improves upon syntax checking with syntax highlighting. Code components like constants, code statements and comments are highlighted in different colors for easy identification. Under VAX/VMS we used the inherent software version control supplied by the VMS file system. Since VMS maintained revisions of source modules until they are explicitly purged, older versions could be easily recovered by deleting the more recent versions. At present the Windows NT file system does not support multiple file versions so we had to purchase a software version control utility. The version control software offered many more features than we required but some of these features prove to be quite useful. Rather than just keeping the last few revisions, the software version control keeps all version and can restore any previous version upon request. Although we had access to the VAX/VMS code profiler, we never used it because of the extra effort required. The Windows NT code profiler is enabled with a simple dialog check box. The ease of use makes the tool much more valuable in determining potential efficiency problems.

3. PORTING CODE

Most of our code compiled under Windows NT without problems but there were a few areas that required some effort to port. These include cursor control, graphical drawing and the network interface. At present we have ported approximately half of the operator interface code to Windows NT. Since our code relies on some non-standard GUI features we had to investigate ways to bypass some of the Windows NT GUI features. Fortunately Windows NT has available all the hooks needed by our software and after a little experimenting we were able to accomplish our tasks. In addition, we had previously used our own network protocol for communication but it was simpler to use UDP/IP that is supported by Windows NT than to add our network protocol to Windows NT.

3.1 Existing Code

Many facilities use external knob boxes or GUI-based sliders for virtual analog knob support under their software. We have used another method that I have not seen at any other facility. Our implementation uses the track ball that is normally used for cursor control in GUI environments. This is accomplish by confining the cursor to the main window, hiding the cursor from view and then placing the hidden cursor in the center of the screen. Every tenth of a second the cursor position is checked and changes in positions are translated to increments in the analog control. If the hidden cursor gets too close to a screen edge, the cursor is repositioned to the center of the screen. Standard GUI style guidelines discourage the "grabbing" of the cursor but all GUIs seem to provide hooks for these functions. Both X Windows and Windows NT allow the cursor to be bound to a particular window and to be hidden from view. One potential problem with this method involves child windows gaining control of the cursor when the hidden cursor is positioned over them. Our software does not use any child windows so the problem is avoided.

Our user interface appears much as it did when we used color graphic terminals. We use a blank graphic window to draw all the text with only a few lines highlighting the areas around the text. This simple format allows for very fast drawing requiring only a fraction of a second to draw a whole screen. Cursor handling becomes a little more complicated because all mouse events inside the window have to be interpreted by the program without the normal assistance provided by the GUI environment. This format also allows for very efficient updates of the device readings due to the minimal overhead imposed by directly drawing the text. Many of these speed enhancements were needed for the older, slower workstations but they have become less important for newer systems.

Under VAX/VMS we developed our own network protocol because it was simpler than writing a complete implementation of a standard network protocol on our VME systems. It is not as easy to implement a new network protocol under Windows NT. It appears that the protocols have to be installed as part of the protocol layer of the driver for the ethernet adapter. We did not want to write any code for Windows NT that would have to be installed as part of the operating system. So we had to approach the problem from the VME system end and write an implementation of UDP/IP for the VME systems [2]. Once the UDP/IP implementation was completed for the VME systems it was only the simple matter of using WinSock for Windows NT. WinSock is an extension of the standard UNIX Berkeley Sockets implemented under Windows NT.

3.2 Commercial Libraries

Our existing code contains some two-dimensional plotting routines needed to display plots of storage ring devices versus time. This plotting code would draw the plot in a empty window using only the text and line drawing functions of the GUI. In porting the code to Windows NT it became apparent that there are several low cost plotting packages that could be used for this function. Under VAX/VMS the plotting packages were so expensive we implemented our own simple plotting routines. By using these inexpensive packages we can create many styles of plots that we could not produce with our own simple plotting software.

4. COMMERCIAL SOFTWARE

One of the main disadvantages of using VAX/VMS systems is the lack of inexpensive commercial software packages. DEC did develop a licensing program that allowed educational institutions access to their code for a small annual fee. Unfortunately this did not reduce the cost of software from other commercial developers. For Windows NT we can purchase many commercial applications at a reduced educational price.

4.1 LabView

LabView is a commercial graphical data acquisition and control program developed by National Instruments that runs under many operating systems including Windows NT. The program offers the graphical control displays that are now common for GUI environments. It also has a graphical programming environment that does not require knowledge of traditional programming languages. This graphical programming environment appeals to our scientific and engineering staff as a simple tool that they can manipulate to accomplish many control related tasks without the need of a controls programmer. In practice there is a steep learning curve needed to work effectively in this environment. Even though the graphical programming environment hides the underlying programming language, knowledge of standard programming concepts are needed to develop the graphical code. At present our staff is reluctant to attempt to make their own changes to the graphical code but they should be able to make these changes after they have developed a little more experience.

The graphical programming environment has several disadvantages compared with traditional programming languages. The graphical language is not as complete as a traditional programming language. All working variables have to be created on the graphical control display and then marked as not visible. Simple functions that require only three or four lines in a traditional programming language can require a whole screen of the graphical programming language. The graphical programming language is best suited for simple data manipulation and display. Very impressive graphical displays can be created easily but much of the underlying data manipulation is best implemented in a traditional programming language. Fortunately there are several methods to link a traditional programming language with LabView, the simplest is through dynamic link libraries.

4.2 Dynamic Link Library

Dynamic link libraries (DLLs) are code and/or data that is not part of the original executable. The DLLs are loaded at run-time by the original executable. This method allows commercial software developers to let end users add custom software to the original commercial executable. Parts of our original control code can be made into a DLL and added to any Windows NT application that allows this method of extension.

4.3 Other Packages

LabView was the first Windows NT application that we thought would be useful in complementing our existing control software. Other packages that we could link to include spreadsheets, mathematical packages, and databases. Spreadsheets and many mathematical packages allow the end user to add functions through dynamic link libraries. At present we have not started to develop links to these applications but we will attempt to include them in future developments. We have started to use a Windows NT database to store device descriptions. Our VMS version was very outdated and needed extensive revisions. The entire device description database was moved to a Windows NT database in only a few days.

5. CONCLUSION

Changing operating systems and windowing systems involves many modifications to the original software. Normally we have only made extensive changes when the existing systems or software packages have become obsolete. The VMS operating system is not obsolete but it is clearly no longer the primary system used at our facility. Windows NT offers a very robust development environment and makes available many inexpensive commercial applications. The initial effort of porting the control software to Windows NT is justified by allowing us to utilize more commercial software. Future software development efforts will be in integrating and extending commercial packages rather than writing our own software.

This work was supported by the National Science Foundation. The current contract number is DMR-9212658.

References

[1] J. P. Stott and D. E. Eisert, CERN Yellow Report **90-08**, 103 (1990)

[2] D. E. Eisert, Fermilab Report, These Proceedings.

Equipment Manager of the VME Control System for the SPring-8 Storage Ring

A. Taketani*, S. Fujiwara, T. Fukui, T. Masuda,
R. Tanaka, T. Wada, W. Xu, and A. Yamashita

SPring-8, Kamigori, Ako-gun, Hyogo 678-12, Japan

We havedeveloped an "Equipment Manager" server process on a VME system to control SPring-8 storage ring equipment. The design concept of the EM is a device abstraction that is to control accelerator components with a minimum knowledge of the low-level physical device configuration. It is close to the object-oriented concept, where the accelerator equipment can be handled as objects.

1. Introduction

In the control system of the SPring-8 storage ring, workstations are used for the operator consoles and VME systems control each accelerator equipment. They are connected by an optical Ethernet and a FDDI network. The general description of the control system is presented in this conference [1].

A client-server model between high level and low level control can produce a flexible software structure. It means multiple processes can run asynchronously and be connected through a small number of Application Program Interfaces (API). We haveadopted the client-server model for access to the accelerator equipment. The equipment management server, called the "Equipment Manager" (EM), is running on the VME-based CPU board under a UNIX-like real-time operating system (HP-RT). The EM drives I/O modules and lower-level controllers through the VME bus. Access to the accelerator components outside the EM is forbidden. The client process, called the "Access Server" (AS), is running on the operator console workstations [1]. The EM and the AS are connected by the ONC/RPC protocol with TCP/IP throughout the network. The AS sends a request to the EM as a function call. Then the result of the request is the return of the value of the function to the AS. A poller for periodical data taking runs on the same CPU board as the EM [1]. It accesses the EM in a similar manner.

2. Equipment access

The hardware address is needed to access every accelerator component through the VME system. It consists of the CPU name on the network, the device name in the local CPU, thechannel number on the VME I/O module and so on. The hardware addresses strongly depend on the configuration of the physical device, but not on that of the logical equipment. We adopted the device abstraction concept, which allows us to use the logical equipment access with a minimal knowledge of the low-level physical device configuration. The logical equipment can correspond to single or multiple physical device(s). The machine operator controls the logical equipment, but not each physical device. The EM resolves commands for a logical equipment and makes linkage(s) to the actual physical device(s). This scheme allows a flexible architecture for the design of application software separated from the complex device dependent part. For example, if a device is replaced the EM must be modified. The client of the EM and higher software can be changed minimally. Fig. 1 shows the logical software position of the EM between logical and physical layers.

We have five major equipment groups: the magnet, the RF, the vacuum, the beam monitoring, and the beamline. None of the HP-RT CPU boards are assigned to multiple equipment groups. The physical device structure of a single VME system is easy to implement in the logical structure.

3. Command and response

When the EM receives a logical command which consists of a character string from the AS/Poller, it translates it into commands for the physical devices. The physical device returns a result that is a binary string. The EM converts this to a logical response and replies to the AS/Poller. The logical command and response are described in an English-like syntax S/V/O/C, where the S is a sender's identification, the V is an action, the O is a target equipment, and the C is a value to be written or the status to be read. The elements are separated by the '/'. This message syntax is close to the object-oriented concept, which is quite easy to understand for accelerator operators. The S/V/O/C command syntax is widely used in the control system of the SPring-8 storage ring.

The S includes a host name, a process identification, an application name, and the owner of the process. It is planned to be used in the EM for security and access control.

The V is a simple word for the action. Currently "put" and "get" are supported in the EM. The former sends the requested value to the equipment. The later gets the value from the equipment.

The O is the most significant term for the device abstraction concept. It indicates a unique logical equipment without complicated hardware address information. Its naming structure is determined by the configuration of the logical equipment. For example, the power supply of a bending magnet in the storage ring is abstracted as "sr_mag_ps_B". The first steering magnet in the second cell is described as "sr_mag_ps_ST_2_1".

The C is the value to send to or the reply from an equipment. Only a logical value is used instead of the digital bit string of the physical layer. For instance, the current of a power supply should be expressed as "10.5A". The EM has the ability to convert from logical to physical values and vice versa.

4. Generic EM structure

The software structure of the EM is divided into two parts. One is the common part for all the equipment operation that is written by the control group. The other is the equipment specific.

The common part handles the interface to the client process of the EM. It resolves the logical commands by reference to a configuration table. The resolved result is passed to the equipment specific part. The low level response is translated to a logical one by a table. The configuration table includes the hardware address information and constants for the conversions. This table is created by a process on the workstation and downloaded to the EM at initialization.

The equipment group must supply three functions for each equipment and describe a configuration table as follows:
(1) a translation from logical C to physical value,
(2) a VME control function which calls the device driver,
(3) a translation from physical value to logical C.
The logical command is stored with the corresponding reference to the three functions with their arguments in the configuration table.

An example of a configuration table is shown in table 1.

When a physical device or its access scheme is replaced, the corresponded functions and the table must be modified.

Table 1: An example of a configuration table. The first line gives the V and O combination for the abstracted command to a magnet power supply. This is followed by the conversions to hardware addressesetc. for three actions the C may call - swirch on, swirch off and set value to one ampere.

```
put/sr_mag_ps_ST_1_1
  on        em_mag_st_on /dev/rio_dev0 1
            none
            em_std_ret
```

```
off     em_mag_st_off /dev/rio_dev0 1
        none
        em_std_ret
%1.0A   em_mag_st_current_put /dev/rio_dev0 1 1 1
        em_mag_st_calib_put  1 6.5535e+3 3.27675e+4
        em_std_ret
```

5. Research and development

While designing the EM, we have been discussing the structured design concept. A CASE tool [4] was used for the structured analysis of the common part of the EM.

The common part of the EM has been developed and tested in a HP-UX environment for rapid development. Most of the EM functions do not require the real-time extensions of UNIX. First we implemented the EM for a steering magnet power supply. A software simulator for the power supply was used instead of the real device in the early stages. The EM with the simulated power supply was a great help in the development. After the test on HP-UX, the EM was ported to HP-RT with a few simple modifications. The EM without the physical device and RPC takes about 400 µs to handle an abstracted command, as measured while using a 742rt/50MHz CPU board.

6. Status

We implemented the lower-level software [2] in the EM instead of the simulator to control the real equipment. The lower-level software include device drivers and their dedicated libraries to drive the physical devices. The RIO system, which is a kind of field bus, was adopted for magnet control [2][3]. We used two steering magnets with their power supplies for the test. This was adequate for the test because it included most of the essential pieces for equipment control by the EM. In the test arrangement the EM is controlled by the AS [1] on a workstation through RPC. The EM on the 743rt/64MHz CPU board drives a RIO master module on the VME bus. The RIO master sends commands to the RIO slaves that are attached to the power supplies. The tests were successful in allowing the remote operation of the power supplies - switching on and off, setting and reading current, reading status, and resetting errors remotely.

7. Reference

[1] R. Tanaka, et al., these Proceedings.
[2] T. Masuda, et al., these Proceedings.
[3] H. Takebe, et al., Proc. of the 4th European Particle Accelerator Conf., London, June 1994, Page 1827-1829.
[4] Teamwork by Cadre Technologies Inc.

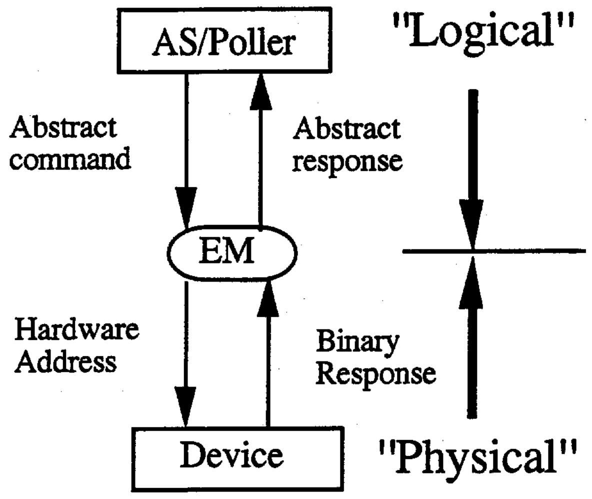


Fig. 1: The software boundary condition of the EM between the logical and physical layer. The EM exchanges the abstract message with the AS and the Poller. The physical device is controlled with the hardware address and binary value.